UNIVERSIDAD CARLOS III DE MADRID

ESCUELA POLITÉCNICA SUPERIOR

DEPARTAMENTO DE TELEMÁTICA

INGENIERÍA DE TELECOMUNICACIÓN

PROYECTO FIN DE CARRERA

Optimización de Comunicaciones TCP a través de canales de transmisión de banda estrecha

AUTORA: ROSANA ZARAPUZ PUERTAS
TUTOR UC3M: MARIO MUÑOZ ORGANERO

Índice de contenidos

1. OBJETIVOS DEL DOCUMENTO 9
1.1. INTRODUCCIÓN 9
1.2. OBJETIVOS DEL DOCUMENTO 9
1.3. ESTRUCTURA DEL DOCUMENTO 10
2. ANÁLISIS DEL PROTOCOLO 13
2.1. INTRODUCCIÓN 13
2.2. VISIÓN GENERAL 13
2.3. INFORMACIÓN ESPECÍFICA PARA EL PROYECTO 15
2.3.1. Inicio de conexión TCP 16
2.3.2. Envío de datos 17
2.3.3. Cierre de conexión 18
2.3.4. Ventana de congestión 18
3. DESCRIPCIÓN DEL SISTEMA COMPLETO 21
3.1. INTRODUCCIÓN 21
3.2. VISIÓN GENERAL 21
3.3. ARQUITECTURA 22
4. DESCRIPCIÓN DEL ALGORITMO 27
4.1. INTRODUCCIÓN 27
4.2. VISIÓN GENERAL 27
4.3. INTERCAMBIO DE DATOS 30
4.4. PASOS DE UNA CONEXIÓN 31
4.4.1. Establecimiento de conexión: Aplicación - AdO 31
4.4.2. Envío de Datos Aplicación - AdO 32
4.4.3. Envío de datos AdO - AdO 33
4.4.4. Establecimiento de conexión AdO - Aplicación 36
4.4.5. Envío de datos AdO - Aplicación 38
4.4.6. Cierre de conexión 40
4.5. CASOS DE INTERÉS 42
4.5.1. Tratamiento de Reset 42
4.5.2. Limitación de capacidad 44
4.5.3. Configuración 45
4.5.4. Temporizadores e interrupciones 46
4.5.5. Comunicación AdO – AdO: Resumen 47
5. MEMORIA TÉCNICA 51
5.1. INTRODUCCIÓN 51
5.2. ESTRUCTURAS DE DATOS 51
5.2.1. Estructura de intercambio 51
5.2.2. Lista de conexiones 52
5.2.3. Variables y constantes 54
5.3. FUNCIONES 55
5.3.1. CheckPktTCPCapturador 56
5.3.2. CheckPktTCPAdaptador 63

5.3.3. CheckAckAdaptador 68
5.3.4. MergeBufferCapturador 69
5.3.5. MergeBufferAdaptador 70
5.3.6. getNumSecuence 71
5.3.7. sendBuffer2Capturador 72
5.3.8. sendBuffer2Redirector 73
5.3.9. SendPkt 74
5.3.10. NewConectionCapturador /NewConectionAdaptador 74
5.3.11. Funciones de búsqueda y retransmisión 75
5.3.12. Funciones de Borrado 75
5.3.13. Funciones de creación de paquetes 76
5.3.14. Interrupciones 77
5.4. ANÁLISIS DE RESULTADOS 79

6. MEMORIA ECONÓMICA 84

6.1. COSTE DE MATERIALES. 84
6.2. COSTE DE PERSONAL. 85
6.3. COSTE TOTAL DE EJECUCIÓN. 85

7. CONCLUSIONES Y LÍNEAS FUTURAS 88

7.1. CONCLUSIONES. 88
7.2. MEJORAS FUTURAS. 88

8. ANEXOS 92

8.1. GLOSARIO DE TÉRMINOS 92
8.2. ESTRUCTURA DE INTERCAMBIO 93
8.3. PRESENTACIÓN 95
8.4. CÓDIGO IMPLEMENTADO 113

9. BIBLIOGRAFÍA 158

Índice de Figuras

Figura 1: Formato de cabecera TCP *13*
Figura 2: Diagrama de flujo básico *14*
Figura 3: Inicio de conexión *16*
Figura 4: Diagrama de bloques del Servidor de Comunicaciones *22*
Figura 5: Escenario. Ejemplo 1 ..... *28*
Figura 6 Escenario. Ejemplo 2 ..... *29*
Figura 7 Estructura de Datos *30*
Figura 8: Establecimiento de conexión *31*
Figura 9 Establecimiento de conexión: Vencimiento de timer *32*
Figura 10: Envío de datos Aplicación – AdO *32*
Figura 11: Pérdida de paquetes Aplicación AdO *33*
Figura 12 Envío de datos AdO – AdO *34*
Figura 13: Envío de datos AdO- AdO. Vencimiento del timer *34*
Figura 14 Envío de datos AdO – AdO. Retransmisión *36*
Figura 15 Establecimiento de Conexión AdO - Aplicación *36*
Figura 16 Establecimiento de Conexión AdO – Aplicación. Retransmisión *37*
Figura 17 Establecimiento de Conexión AdO – Aplicación *37*
Figura 18: Envío de datos AdO – Aplicación *38*
Figura 19 Envío de datos AdO – Aplicación. Pérdida de paquetes *39*
Figura 20 Confirmación de datos AdO – AdO *39*
Figura 21 Cierre de conexión 1 extremo *41*
Figura 22 Cierre de conexión 2 extremos *42*
Figura 23 Tratamiento de Reset *43*
Figura 24 Tratamiento de Reset (2) *44*
Figura 25 Limitación por capacidad *45*
Figura 26 Estructura headerPacket *51*
Figura 27 Estructura de Datos *52*
Figura 28 Diagrama de Flujo. CheckPktTCPCapturador *56*
Figura 29 Diagrama de Estados. CheckPktTCPCapturador *60*
Figura 30 Diagrama de Flujo. CheckPktTCPAdaptador *63*
Figura 31 Diagrama de Estado. CheckPktTCPAdaptador *66*
Figura 32 Diagrama de Flujo. CheckAckAdaptador *68*
Figura 33 Diagrama de Flujo. MergeBufferCapturador *69*
Figura 34 Diagrama de Flujo. MergeBufferAdaptador *70*
Figura 35 Diagrama de Flujo. getNumSecuence *71*
Figura 36 Diagrama de Flujo. sendBuffer2Capturador *72*
Figura 37 Diagrama de Flujo. sendBuffer2Redirector *73*
Figura 38 Escenario de Pruebas *79*

CAPÍTULO 1:
Objetivos del documento

1. Objetivos del Documento

1.1. Introducción

El presente documento describe el diseño, desarrollo y puesta en marcha de un algoritmo dentro de un servidor de comunicaciones que permite establecer comunicaciones TCP en redes de banda estrecha siguiendo unas especificaciones dadas.

El producto *software*, que constituye el proyecto fin de carrera, ha sido desarrollado para formar parte, como módulo, de un sistema mayor denominado Servidor de Comunicaciones, y "**desarrolla un algoritmo que permite establecer conexiones TCP entre dos nodos a través de canales con un alto retardo, o una muy baja velocidad de transmisión**", que debido a las características del protocolo TCP no completarían el inicio de sesión.

1.2. Objetivos del documento

Este documento explica el funcionamiento de lo que se ha denominado "**Algoritmo de Optimización**", en adelante AdO, mostrando su funcionamiento; incluye una descripción técnica del esquema global, profundizando en aquellas partes más relevantes.

El desarrollo software se ha implementado en el lenguaje de programación C/C++ estándar. Para ilustrar el funcionamiento general y explicar el intercambio de mensajes con otros módulos constitutivos del Servidor de Comunicaciones, se incluyen diversos diagramas temporales, así como diagramas de flujo para describir la estructura del código.

El documento también incluye una visión general del protocolo TCP, para poder conocer los aspectos principales del funcionamiento de dicho protocolo y entender mejor las características que se pretenden mejorar, así como una visión global del producto completo, para poder situar el algoritmo desarrollado en su marco adecuado.

1.3. Estructura del documento

Vamos a describir la temática de cada uno de los capítulos en los que se ha organizado este proyecto:

- En primer lugar está el índice del documento.
- El presente capítulo constituye una descripción de los objetivos del documento y la estructuración del mismo.
- Capítulo 2 "**Análisis del protocolo**": Da una visión general del protocolo TCP y una explicación de sus aspectos más relevantes de cara a este proyecto.
- Capitulo 3 "**Descripción del sistema Servidor de Comunicaciones completo"**: Muestra una visión de todo el sistema en el que se incluye el algoritmo que se ha desarrollado como un bloque más y una breve explicación de los diferentes módulos que componen el sistema.
- Capítulo 4 "**Descripción del Algoritmo**": pretende explicar el funcionamiento del algoritmo así cómo dar al lector una visión global de dicho algoritmo.
- Capítulo 5 "**Memoria Técnica**": Incluye los diagramas de flujo y las principales funciones del programa, así como las pruebas realizadas y el análisis de resultados.
- Capítulo 6 "**Memoria Económica y presupuesto**": En este capítulo se realiza una valoración del trabajo realizado para la realización del proyecto.
- Capítulo 7 "**Conclusiones y líneas futuras**": Se incluye la valoración de los objetivos del algoritmo así cómo posibles mejoras.
- Un anexo con información complementaria.
- Un anexo con la presentación usada en la exposición del proyecto.
- El código implementado como anexo.
- Por último una bibliografía con las distintas fuentes de información utilizadas a lo largo de este proyecto.

CAPÍTULO 2:
Análisis del protocolo

2. Análisis del protocolo

2.1. Introducción

Este capítulo no pretende ser una explicación teórica del protocolo TCP, sino simplemente mostrar parámetros y variables de interés en el funcionamiento de dicho protocolo, profundizando en aquellos aspectos que más han influido en el desarrollo del algoritmo sobre el que versa este proyecto.

Si el lector deseara obtener una información completa y más detallada sobre su funcionamiento, siempre puede consultar la RFC 793 o las referencias incluidas en la bibliografía.

2.2. Visión general

El protocolo TCP es un protocolo orientado a conexión con mucha difusión, que crece según se dispone de infraestructura con mayor ancho de banda. La estructura de su cabecera es la mostrada en la figura.

Figura 1: Formato de cabecera TCP

El protocolo TCP proporciona una cantidad considerablemente mayor de servicios a las aplicaciones que el protocolo UDP no presta, protocolo usado para aplicaciones en tiempo real, en especial:

- La recuperación de errores.
- El control de flujo.
- La fiabilidad. Se trata de un protocolo *orientado a conexión* a diferencia de UDP.

La mayor parte de las aplicaciones que se desarrollan hoy en día se diseñan con el uso de este protocolo, dado que se parte del hecho de que no va a existir problemas de retardo al utilizarse redes de banda ancha.

Este protocolo exige que el extremo receptor devuelva al extremo emisor un paquete o trama de datos con información de lo que ha recibido hasta ese momento, para que el extremo emisor continúe mandando información, como se indica en el grafo de la figura:

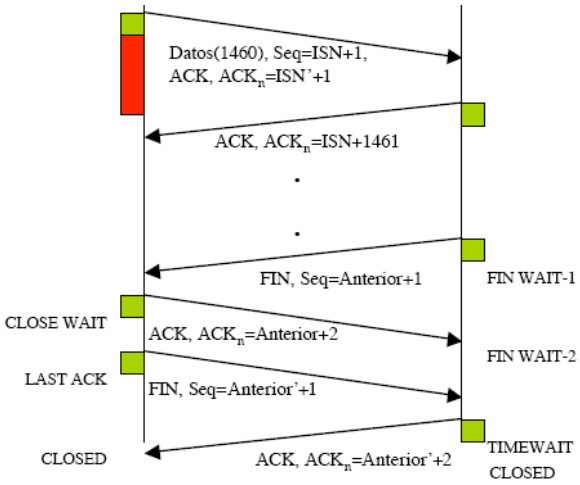


Figura 2: Diagrama de flujo básico

La cantidad de información mandada hasta exigir una respuesta, viene definido por el "Tamaño de la Ventana".

Para el cálculo del tiempo de espera a la respuesta, TCP registra el momento de envío de un segmento y el de recepción del paquete de confirmación ACK. Se promedia un valor para varios de estos viajes, que se empleará como valor de *TimeOut* para el siguiente segmento a enviar. Es decir, la adaptación de los temporizadores se basa en estimar el tiempo de "ida y vuelta" *(Round Trip Time-RTT*). Este RTT se actualiza en función del tiempo obtenido en el último segmento y el RTT en momentos anteriores, calculándose el *TimeOut* en función de este RTT (generalmente, el doble).

En las redes de banda estrecha es muy posible que la respuesta no llegue a tiempo. Bajo esta circunstancia, el vencimiento de los temporizadores de retransmisión de los segmentos agrava considerablemente el problema de utilizar este protocolo en una red de Banda Estrecha.

En caso de pérdida de un segmento, es decir, vencimiento de *TimeOut*, el RTT obtenido no se tiene en cuenta, y para la retransmisión se duplica el *TimeOut*.

En el tipo de redes de Banda Estrecha no se puede aplicar la técnica de "la ventana de congestión" que consiste en aproximarse al límite a la ventana máxima de transmisión. Esta ventana, que inicialmente sólo permite la transmisión de un segmento, aumenta el tamaño de la ventana de congestión en un segmento por cada segmento transmitido con éxito (reconocido sin necesidad de reenviarlo). Cuando se alcanza la

mitad de la ventana de transmisión concedida por el receptor, únicamente se incrementa en uno la ventana cuando todos los segmentos de la misma han llegado sin problemas a su destino. Y de forma complementaria, cuando se produce una retransmisión, se considera que la red puede estar congestionada, y por tanto entra en acción el denominado "*multiplicative decrease*", dividiendo por dos la ventana de congestión.

Se puede producir con frecuencia el cierre abrupto. En estos casos, no es posible continuar con la transferencia TCP. Se emplea un único segmento con el bit RST (Reset) activo y se detiene la comunicación TCP. RST no garantiza la entrega de todos los datos transferidos, eliminando instantáneamente la conexión y sus ventanas asociadas.

2.3. Información específica para el proyecto

Se va a proceder a explicar el significado de algunos campos de la cabecera TCP así como su funcionamiento práctico, destacando aquellos que se emplearán en el Algoritmo de Optimización.

Tal y como muestra la Figura 1, los diferentes campos de la cabecera de un segmento TCP son:

- Los campos *puerto origen* (16 bits) y *puerto destino* (16 bits) identifican la aplicación emisora y receptora respectivamente.
- El campo *número de secuencia* (32 bits) indica el número de orden relativo del primer octeto de datos de este segmento con respecto al resto de datos enviados de otros segmentos.
- El *número de confirmación* (32 bits), o número de acuse de recibo, indica el valor del siguiente número de secuencia que el emisor de este segmento espera recibir. El valor de este campo sólo es válido cuando el bit de control de ACK está activo (tiene el valor de 1).
- El campo *Longitud de Cabecera* (4 bits) indica dónde comienzan los datos. Es el número de palabras de 32 bits contenidas en la cabecera TCP. Como existe un campo *Opciones* cuya longitud es variable se hace necesaria esta información.
- Los siguientes 6 bits están reservados para uso futuro.
- Los 6 *bits de control* se dividen en:
    - URG: indica la validez del campo *Puntero de urgencia*.
    - ACK: indica la validez del campo *Número de confirmación*.
    - PSH: Si está activo, solicita al receptor entregar los datos a la aplicación inmediatamente, y no almacenarlos hasta la recepción de un buffer completo.
    - RST: Reinicia ('Reset') la conexión.
    - SYN: Sincroniza los números de secuencia. Se emplea al establecer conexiones.
    - FIN: indica los últimos datos del emisor, se emplea para liberar una conexión.

  Estos bits se consideran activos cuando su valor es 1.

- El *tamaño de la ventana* indica el número de bytes a contar a partir del número indicado en el campo *Número de confirmación* que se esta dispuesto a aceptar. Se emplea para el control de flujo.
- La *suma de comprobación* se emplea para proporcionar una protección frente a errores.
- El *puntero de urgencia* señala la posición en la que comienzan los datos urgentes. Sólo es significativo si el bit URG está activo.
- El campo *opciones* se emplea en el establecimiento de la conexión para fijar características como el tamaño máximo de segmento o la retransmisión selectiva.

Los campos que utiliza el Algoritmo de Optimización propuesto son *número de secuencia, número de confirmación, tamaño de ventana* y los bits de control ACK, RST, SYN y FIN.

Los escenarios donde va a trabajar el Algoritmo que se presenta son el establecimiento de conexión, envío de datos y cierre de conexión, así como el tratamiento de reset y limitación de flujo, por lo que se va a dar una visión general de su funcionamiento para entender la función del algoritmo dentro de cada escenario.

2.3.1. Inicio de conexión TCP

El establecimiento de la conexión se conoce como negociación en tres pasos (*three-way handshake*). Normalmente, este procedimiento se inicia por una entidad TCP, aunque también funciona si dos entidades TCP inician el procedimiento simultáneamente.

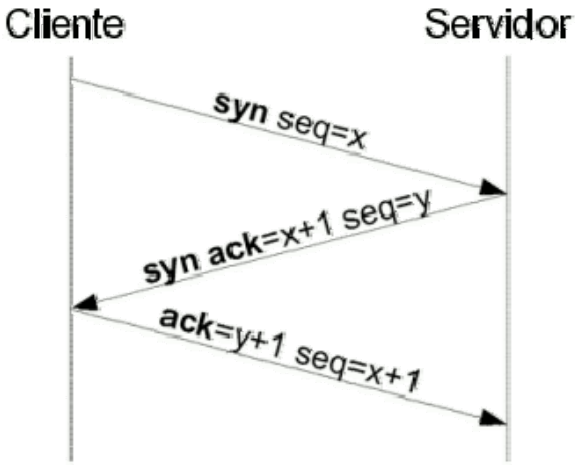


Figura 3: Inicio de conexión

La aplicación que desea iniciar la conexión (“cliente”) envía un segmento TCP a la aplicación destino (“servidor”) con el bit de control SYN puesto a 1, el bit de control ACK a 0 y un número de secuencia inicial elegido por la aplicación (x). Si el servidor acepta la conexión, le envía al cliente un segmento TCP con el bit de control SYN a 1, el bit de control ACK a 1, un número de secuencia inicial elegido por el servidor (y) y

como número de confirmación el número de secuencia enviado por el cliente incrementado en 1 (x+1). El cliente le confirma al servidor que ha recibido su respuesta con un segmento con el bit de control ACK a 1, como número de secuencia su valor inicial incrementado en 1 (x+1) y como número de confirmación el número de secuencia enviado por el servidor incrementado en 1 (y+1). Este último segmento no consume número de secuencia, por lo que el primer paquete de datos que se envíen llevará como número de secuencia x+1 en el caso de que sea el primer paquete de datos del cliente o y+1 si se trata del primer paquete de datos que envía el servidor.

Tras la secuencia SYN, SYN-ACK, ACK, la conexión está establecida por lo que ya se puede iniciar el intercambio de datos entre ambas entidades.

2.3.2. Envío de datos

Una vez establecida la conexión se pueden transmitir datos mediante el intercambio de segmentos. El incremento de los números de secuencia y número de confirmación se realiza en función del tamaño de los datos contenidos en cada segmento, es decir, si el cliente envía un segmento con número de secuencia "x" y longitud de datos de "n" bytes, el servidor, si no hay problemas en la comunicación, le enviará un segmento con un número de confirmación igual a "x+n", el siguiente segmento de datos que envíe el cliente llevará un número de secuencia de "x+n".

Cada uno de los dos extremos de la transmisión tiene constancia de los datos que ha transmitido y recibido, gracias a los números de secuencia y confirmación. Para que un segmento recibido se considere válido se tienen que cumplir ciertas restricciones con los bits de control y los números de secuencia y confirmación. Aquí solo se van a mencionar los casos más generales.

Cuando un extremo de la conexión recibe un segmento con datos, comprueba que el número de secuencia de dicho segmento coincida con el que se espera recibir, si es así el segmento es correcto. En el caso de que dicho número sea menor que el esperado indica que contiene datos que ya han sido recibidos (pueden producirse paquetes duplicados debido a la red o a la retransmisión TCP), el segmento se ignora, pero si el número de secuencia del segmento es mayor que el esperado significa que hay unos datos que el emisor ha enviado pero no han llegado al receptor (puede deberse a perdidas en la red o a retrasos por la diferencia de caminos), por lo que el receptor envía un segmento con el número de confirmación igual al valor esperado. El extremo emisor, si recibe tres segmentos con igual número de confirmación vuelve a retransmitir los datos correspondientes.

2.3.3. Cierre de conexión

El cierre de conexión por un extremo de la comunicación se realiza enviando un segmento con el bit de control FIN a 1. Con esto le indica al otro extremo que no tiene más datos que enviar, aunque puede seguir recibiendo datos del otro extremo (y por tanto mandando la confirmación).

Cuando ambos extremos hayan enviado su segmento de FIN y haya sido confirmado, se inicia un temporizador (igual al tiempo de vida máximo del paquete) para garantizar que no hay paquetes de la conexión circulando por la red (duplicados o retransmisiones). Al expirar el temporizador, la conexión está cerrada.

2.3.4. Ventana de congestión

Para realizar el control de flujo en TCP se emplea el campo de la cabecera "Tamaño de ventana". Dicho campo indica el número de octetos de datos, a contar a partir del número indicado en el campo de "Número de confirmación", que el emisor de este segmento está dispuesto a aceptar. Cuando dicho campo es cero, no se debe aceptar ningún segmento excepto los ACK, y tratar los campos RST y URG de todos los segmentos entrantes.

Si el TCP receptor tiene una ventana de tamaño 0 y le llega un segmento debe contestarlo con un segmento ACK indicando su número de secuencia esperado y el tamaño de su ventana actual. El TCP emisor debe retransmitir al TCP receptor para comprobar el tamaño de la ventana. Cuando el TCP receptor pueda recibir más datos, enviará un segmento ACK con un tamaño de ventana mayor que cero.

CAPÍTULO 3:
Descripción del Proyecto Global. El Sistema de Comunicaciones

3. Descripción del sistema completo

3.1. Introducción

El algoritmo objeto de este proyecto forma parte de un módulo que está incluido en un producto mayor, que se ha denominado Servidor De Comunicaciones, en adelante SC. A continuación se va a tratar de explicar en que consiste dicho SC así como los módulos que lo componen.

3.2. Visión general

El SC es un gestor de encaminamiento y uso alternativo de los medios de transmisión existentes en nodos de comunicaciones, dotados de un conjunto heterogéneo de medios de transmisión, susceptibles de ser gestionados por un operador en función de las necesidades y de su disponibilidad en un determinado momento.

El SC es un servicio del nivel de red (nivel 3 en la jerarquía OSI) cuyo objetivo es el encaminamiento de paquetes IP a través de diferentes redes de transmisión tipo WAN en función de la operatividad de las mismas y de unas reglas de uso preestablecidas.

El encaminamiento de los datos se realiza en función de:

- Derechos de salida y listas de control de acceso
- Redes existentes y su prioridad de uso
- Estado de la conexión de cada una de las redes
- Parámetros y protocolos específicos de cada una de estas redes

Las redes a utilizar: IP-ETHERNET, INMARSAT, GSM/GPRS, TETRAPOL, TETRA, THURAYA, VHF y HF.

Para las redes de bajo ancho de banda y alta latencia se implementan varios mecanismos para optimizar las comunicaciones, entre las que se encuentra la adaptación a nivel de transporte de las sesiones TCP generadas por las aplicaciones de la red de área local, que quieran establecer este tipo de conexiones a través de las redes de transmisión WAN.

Este mecanismo de adaptación TCP permite la optimización de este tipo de conexiones sobre las redes de transmisión de bajo ancho de banda, ya que, debido al retardo que presentan las transmisiones de los paquetes en condiciones normales, se rebasan los tiempos de espera máximos y no se consigue establecer la conexión.

Este es el objetivo principal del Algoritmo sobre el que versa este proyecto: **Que las aplicaciones que utilizan tramas TCP puedan ser transmitidas por canales de banda estrecha.**

3.3. Arquitectura

La aplicación se divide en diferentes módulos o componentes que realizan tareas específicas:

1. **Supervisor**: permite que se establezcan los parámetros de configuración en la Base de Datos que se aplicarán a los datagramas, así como a las líneas de entrada/salida que controla el sistema.
2. **Capturador**: es el encargado de la captura del tráfico no local para su envío a la WAN y la entrega de datos procedentes de la WAN a la red LAN.
3. **Discriminador**: es el encargado de aplicar las reglas de acceso y del mantenimiento de conexiones TCP en configuraciones simétricas sobre redes de baja capacidad.
4. **Redirector**: Determina la red de transmisión que será utilizada para transmitir cada paquete IP en función de las reglas de prioridad y estados proporcionados por el supervisor.
5. **Adaptadores**: Invocan los procedimientos específicos de cada driver de comunicaciones y realiza el control de la congestión y tratamiento de colas.
6. **Drivers**: Configuran y manejan los dispositivos físicos de cada canal de transmisión

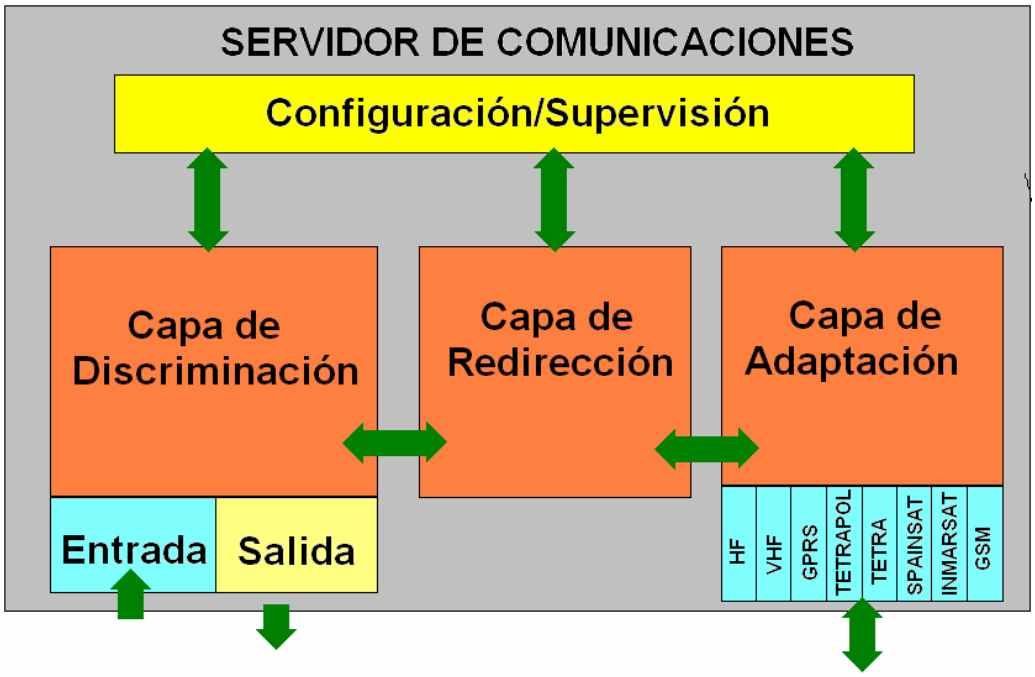


Figura 4: Diagrama de bloques del Servidor de Comunicaciones

Debido a la descomposición en diferentes módulos se necesita un mecanismo de comunicación entre ellos. La transferencia de información entre los diferentes módulos se realiza mediante memorias compartidas. Para sincronizar el acceso a estos recursos compartidos entre los diferentes módulos se emplean semáforos y secciones críticas.

Los módulos que más afectan al algoritmo son el Capturador y el Discriminador, por lo que se va a proceder a explicar un poco más en detalle su funcionamiento:

- Capturador

Se encarga de la captura y escritura de las tramas existentes en LAN. La captura de tráfico se realiza a través de la librería Winpcap 3.1.

WinPcap es una librería gratuita para la captura de paquetes y el análisis de red para plataformas Win32.

Una vez que el Capturador se encuentra configurado correctamente y el Supervisor lo ha activado, el Capturador iniciará el hilo de captura de tramas, realizando la captura de la siguiente manera:

- Se seleccionará el adaptador indicado por el Supervisor.
- Se seleccionará el filtrado que indique el Supervisor, siendo agregado al filtro por defecto del Capturador: filtrado de paquetes IP.

El proceso de captura de tramas se puede dividir en:

- Captura de la trama: se extrae la cabecera Ethernet.
- Se extrae la cabecera IP.
- En función de si el protocolo de transporte es UDP o TCP se extrae la cabecera correspondiente.

Una vez que se han extraído todas las cabeceras, se guardan los datos de las mismas en la estructura de intercambio, así como los datos puros de la trama. La estructura de intercambio que se ha definido se denomina *HeaderPacket*. Dicha estructura será explicada en el Capítulo 4.

El Capturador tiene un hilo de escritura de tramas, el cual cuando se le despierta el evento correspondiente y se le proporciona una estructura de intercambio rellenada adecuadamente, procede a escribir la trama en la red de la siguiente forma:

- Se forma la cabecera Ethernet.
- Se forma la cabecera IP.
- Se forma la cabecera TCP o UDP según proceda.
- Se procede a agregar los datos a la trama y se envía a la red.

- Discriminador

El módulo Discriminador es el que recibe los paquetes de su propia LAN (proporcionados por el Capturador), los analiza según ha sido configurado por el Supervisor y los envía al Redirector para que este elija la red de transmisión correspondiente, se lo pase al Adaptador y se transmita el paquete. El Discriminador

también recibe del Adaptador los paquetes procedentes de la WAN con destino a su LAN, los analiza y se los envía al Capturador para que los envíe a la LAN.

Es en este módulo donde se incluye la optimización TCP, tras detectar que es un paquete TCP y que es una conexión simétrica (el destino del paquete pertenece a una LAN donde está presente esta aplicación, esta información es proporcionada por el Supervisor).

La optimización consistirá en suplantar tanto en origen como en destino la identidad del otro extremo de la conexión, para así evitar que venzan los temporizadores y que se aborte la conexión. Para ello se tratarán los diferentes estados del protocolo TCP así como los mensajes intercambiados para que la aplicación final no sea consciente de la suplantación.

Los paquetes que el Adaptador recibe del Algoritmo de Optimización (AdO) son encapsulados, añadiendo una cabecera UDP y enviados al destino, donde el Adaptador remoto detecta la encapsulación y retira la cabecera añadida, por lo que AdO remoto recibe el paquete tal cual lo creo el AdO origen.

CAPÍTULO 4:
Descripción del Algoritmo de Optimización.

4. Descripción del Algoritmo

4.1. Introducción

En conexiones simétricas sobre redes de baja capacidad es necesario implementar un mecanismo de mantenimiento de las conexiones TCP. Este mecanismo, al cual se ha denominado Algoritmo de Optimización, se basa en una **validación local de las tramas TCP**. Para ello, el SC origen valida el establecimiento de conexión TCP y confirma los datos que recibe del cliente. Una vez que el SC ha recibido los datos se envían por WAN utilizando un protocolo SC-SC definido en el AdO. En el extremo receptor, una vez que el SC ha recibido todos los datos, se suplanta la identidad de la aplicación origen y se inicia la conexión TCP, enviando los datos y tratando sus confirmaciones. Si la aplicación receptora tiene algo que transmitir se actúa de igual forma.

Todos los casos tratados donde interviene el AdO se han gestionado siguiendo el comportamiento que sigue el protocolo TCP según la RFC 793 y la RFC 1122.

Todo esto se detalla en los siguientes apartados.

4.2. Visión general

Una vez dada una visión general del sistema completo, nos centraremos en el algoritmo de optimización (AdO).

Es importante señalar que a dicho algoritmo sólo le llegan paquetes que cumplan los siguientes requisitos:

- Son TCP.
- Simétrico: la subred del otro extremo de la conexión también posee el mismo algoritmo de optimización. Esta información la obtiene a través del módulo Supervisor del SC.
- Sin errores: La comprobación del *checksum* la realiza el módulo del Capturador.

Los pasos básicos en una conexión TCP son “establecimiento de conexión”, “intercambio de datos entre emisor y receptor” y “cierre de conexión”:

- Cuando el AdO recibe una petición de establecimiento de conexión desde una aplicación en su misma subred, suplanta la identidad del destino y contesta a la aplicación para establecer la comunicación.

- Si la aplicación envía paquetes TCP de datos, el AdO los asentirá suplantando nuevamente al destino y los almacenará en un buffer. Una vez que la aplicación haya terminado de enviar todos los paquetes, el AdO origen enviará al AdO destino todos los paquetes almacenados siguiendo un protocolo privado para la

comunicación AdO – AdO. Cuando el AdO destino recibe todos los paquetes, se los envía a la aplicación destino suplantando al origen.

- Cuando uno de los extremos envía una petición de cierre de conexión, se le comunica al otro extremo, cuando se cierra la conexión en ambos extremos, se elimina dicha conexión del AdO.

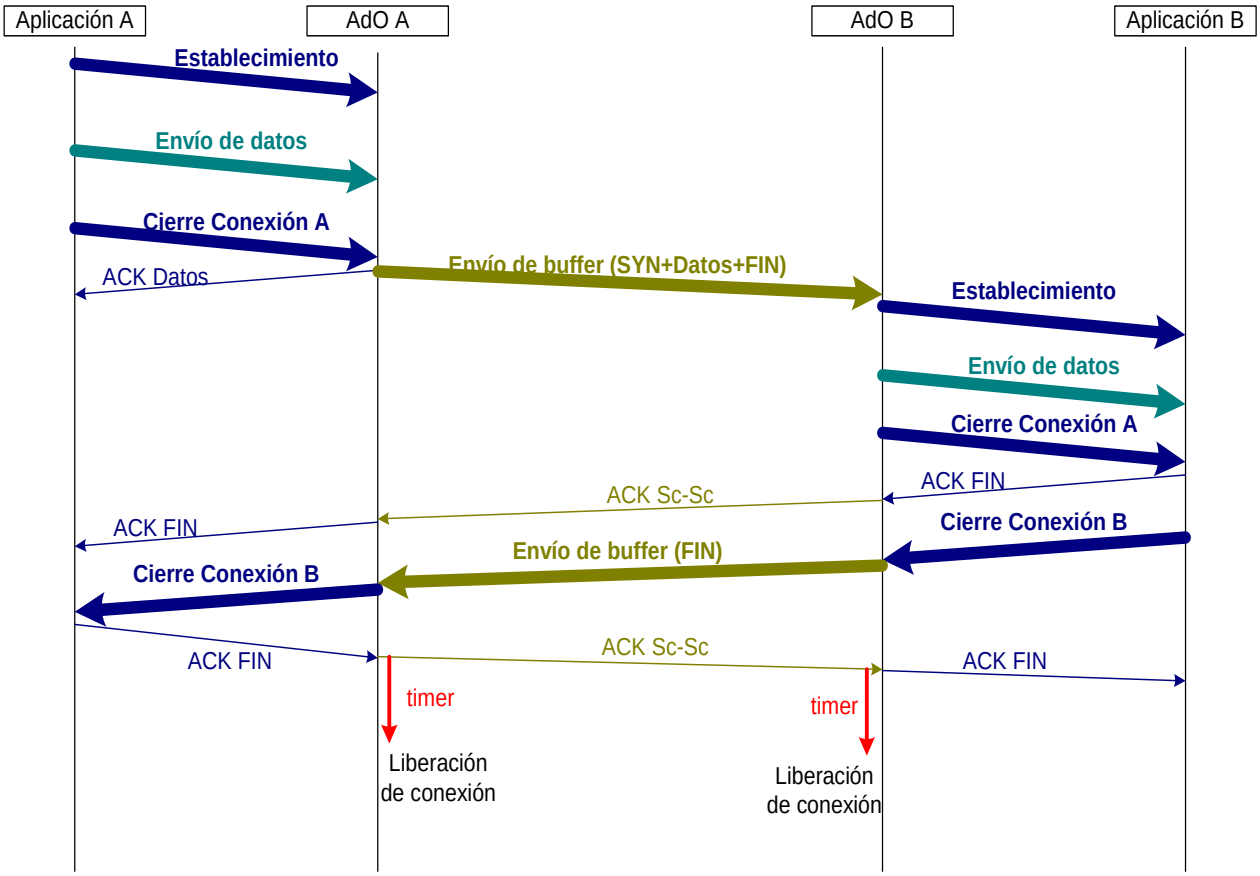


Figura 5: Escenario. Ejemplo 1

La Figura 1 muestra un escenario de ejemplo para ilustrar el algoritmo, que podría corresponderse con el envío de un fichero de la aplicación A a la aplicación B.

La aplicación A establece la conexión con el AdO A, le envía los datos e indica el cierre. El AdO A le envía todos estos datos al AdO B, el cual, cuando los ha recibido todos, inicia el establecimiento de conexión con la aplicación B y le envía todos los datos, incluido el FIN. Cuando la Aplicación B le ha confirmado todos los paquetes, el AdO B se lo notifica al AdO A para que pueda liberar el buffer de memoria. Cuando la Aplicación B inicia el cierre de conexión, se vuelven a comunicar el AdO A y el AdO B para liberar la conexión.

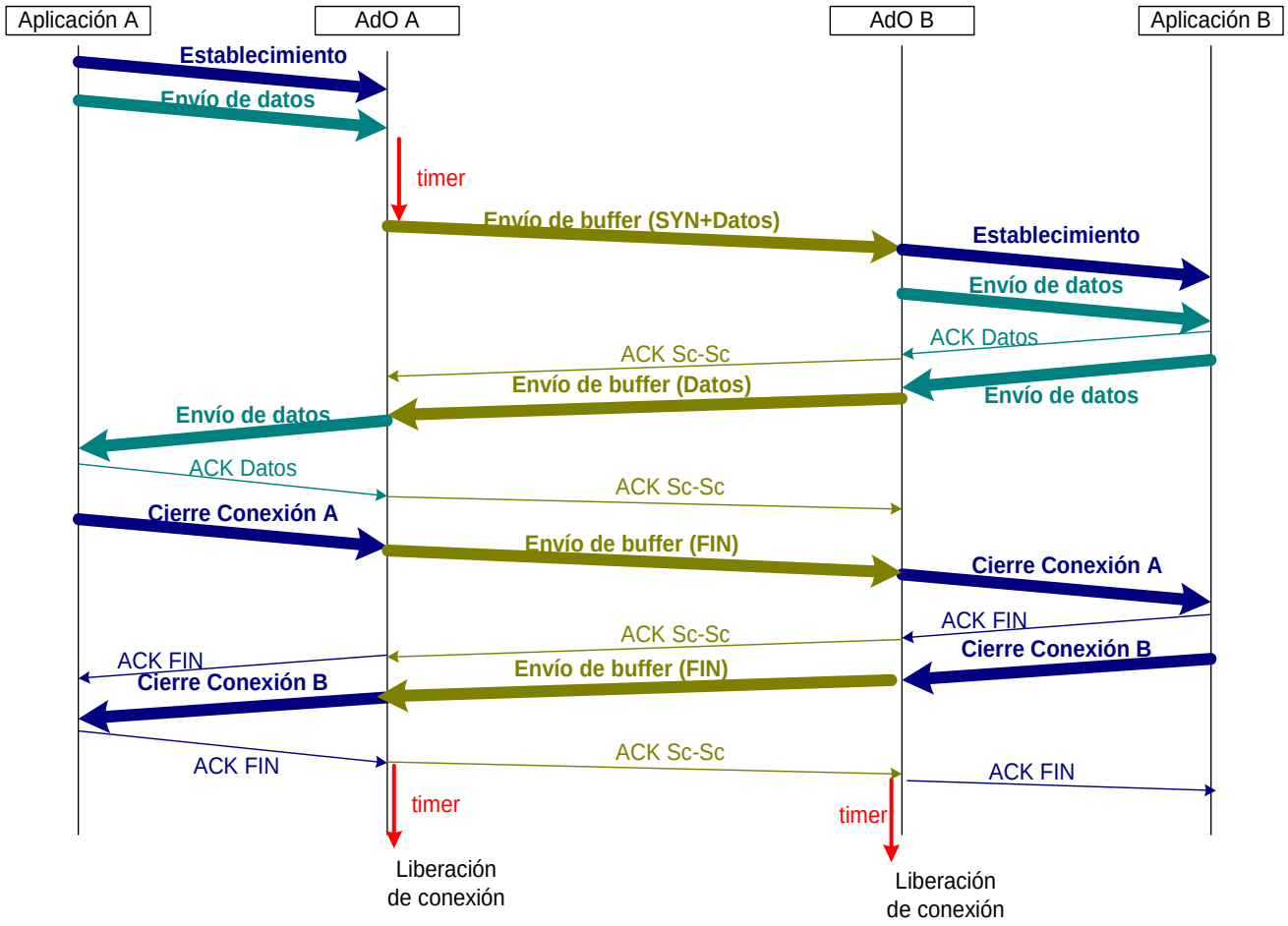


Figura 6 Escenario. Ejemplo 2

El segundo ejemplo muestra otro de los escenarios posibles, que podría asemejarse a un servicio cliente – servidor, donde la aplicación A realiza una consulta a la Aplicación B y espera una respuesta.

La aplicación A establece la conexión con el AdO A, realiza la consulta y se queda esperando, el AdO A, tras esperar un tiempo la llegada de más paquetes, le envía los almacenados en el buffer al AdO B. Tras recibirlos, éste inicia la conexión con la Aplicación B y le envía la petición. La Aplicación B confirma que ha recibido la petición y le envía la respuesta, que el AdO B se encarga de transmitir al AdO A, tras lo cual se cierra la conexión en ambos extremos y se libera.

Estos son sólo dos escenarios que nos sirven para ilustrar el funcionamiento del algoritmo AdO, pero existen muchos más. En ambos ejemplos, el algoritmo es invisible para ambas aplicaciones, la aplicación emisora y el cliente receptor no son conscientes de que hay agentes intermedios. El funcionamiento más detallado de cada uno de los pasos se explicará en los siguientes apartados.

4.3. Intercambio de datos

El intercambio de datos entre los distintos módulos se realiza a través de una estructura denominada *HeaderPacket*, que contiene todos los campos necesarios para trabajar con paquetes recogidos/enviados a la red. El AdO recibe esta estructura cuando corresponde a un paquete TCP en una conexión simétrica, y puede recibirla bien del Capturador (origen LAN) o bien de los adaptadores (destino LAN).

Para manejar estos paquetes se utiliza una lista enlazada que contiene la dirección IP y el puerto Origen de las conexiones activas, y por cada entrada de esa lista se tiene otra lista enlazada para los destinos correspondientes de ese origen. En esta última lista se incluyen los parámetros adicionales necesarios para el procesado por parte del AdO, así como los paquetes recibidos.

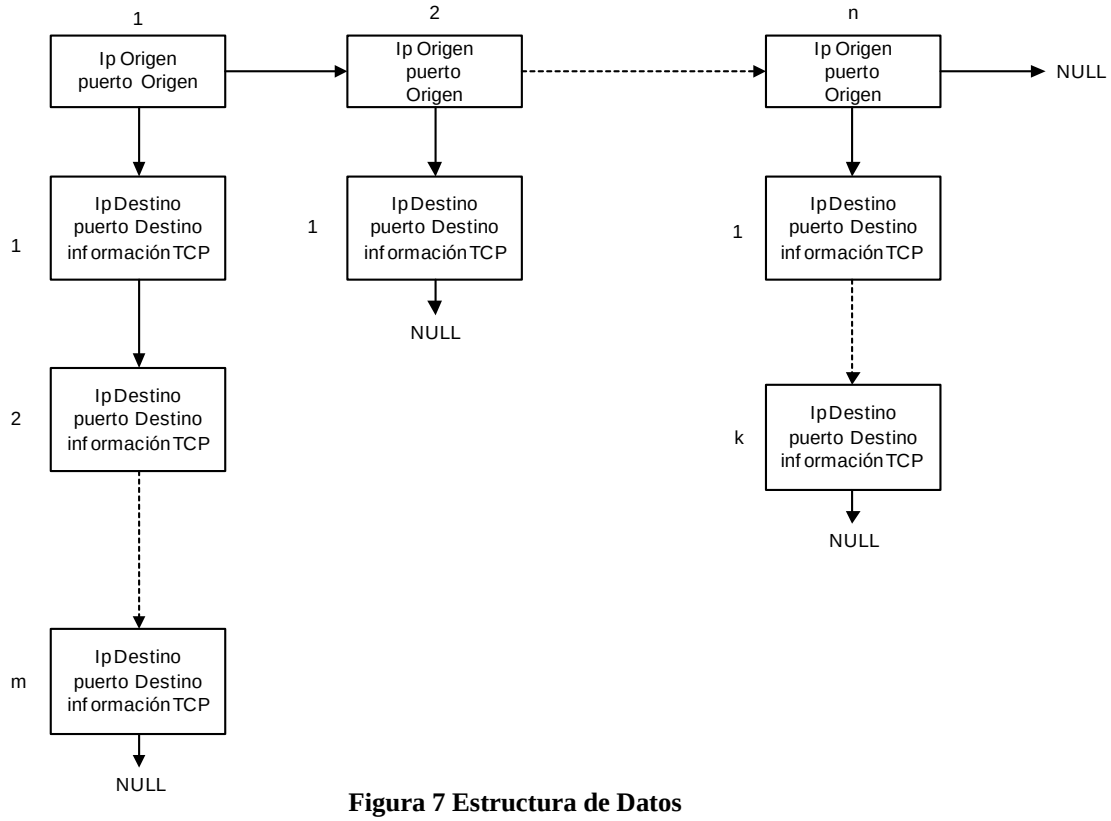


Figura 7 Estructura de Datos

Se trabaja con dos estructuras o listas de este tipo:

1. En la primera se almacenan los paquetes que se reciben del Capturador.
2. La segunda es para los paquetes que se reciben de los Adaptadores.

Una conexión activa tendrá entrada en ambas listas, y existe una referencia cruzada entre ellas para agilizar el acceso. El hecho de emplear dos listas facilita el manejo de los números de secuencia empleados en cada extremo de la conexión, así como el tratamiento de los paquetes recibidos de uno u otro extremo.

En un AdO concreto, una conexión estará **semi-activa** cuando se haya solicitado el establecimiento de conexión pero aún no se haya completado (existirá una entrada para esa conexión sólo en una de las dos listas), y estará **activa** cuando se haya completado, es decir, existirá una entrada para esa conexión en ambas listas.

4.4. Pasos de una conexión

En este apartado se van a detallar los principales pasos que se dan en una conexión TCP entre dos aplicaciones cuando interviene el Algoritmo de Optimización, que son:

1. En la LAN del emisor:
   - Establecimiento de conexión Aplicación – AdO .
   - Envío de datos Aplicación – AdO.
2. Comunicación WAN
   - Envío de datos AdO – AdO.
3. En la LAN del receptor:
   - Establecimiento de conexión AdO – Aplicación.
   - Envío de datos AdO – Aplicación.
4. Comunicación WAN
   - Cierre de conexión.

4.4.1. Establecimiento de conexión: Aplicación - AdO

Cuando el AdO recibe un SYN del Capturador, almacena el paquete y le contesta enviándole un SYN-ACK suplantando la información del destino (dirección IP y puerto) y generando un número aleatorio como número de secuencia inicial. Para completar el establecimiento de la conexión debemos recibir de la Aplicación un ACK confirmando nuestro número de secuencia inicial, el AdO detecta ese ACK y activa dicha conexión, actualizando la información necesaria para su gestión, pero no almacena este último paquete.

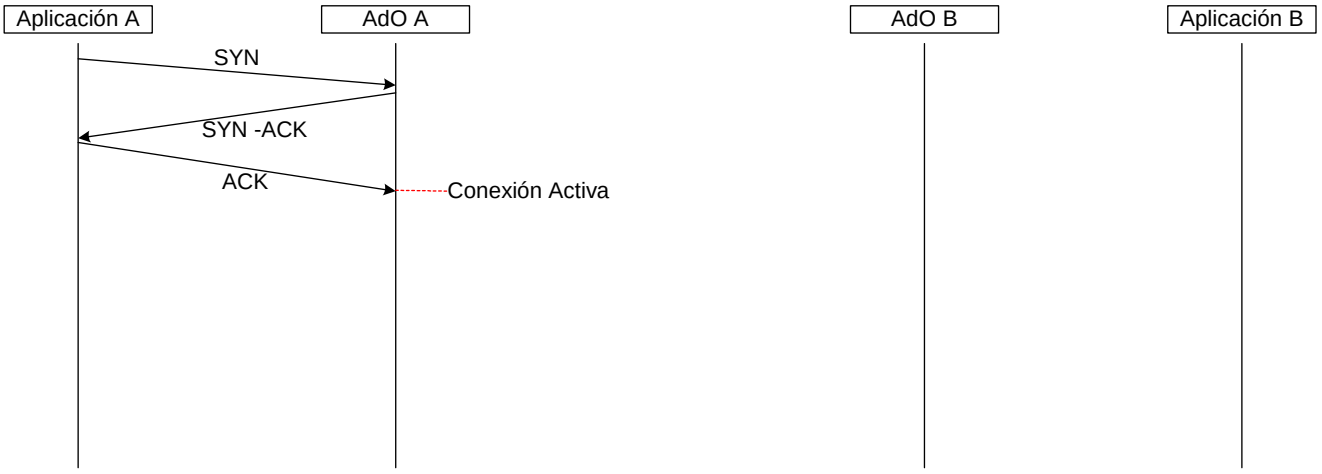


Figura 8: Establecimiento de conexión

Si pasado un tiempo no ha recibido el ACK del establecimiento, se borra la conexión de la memoria.

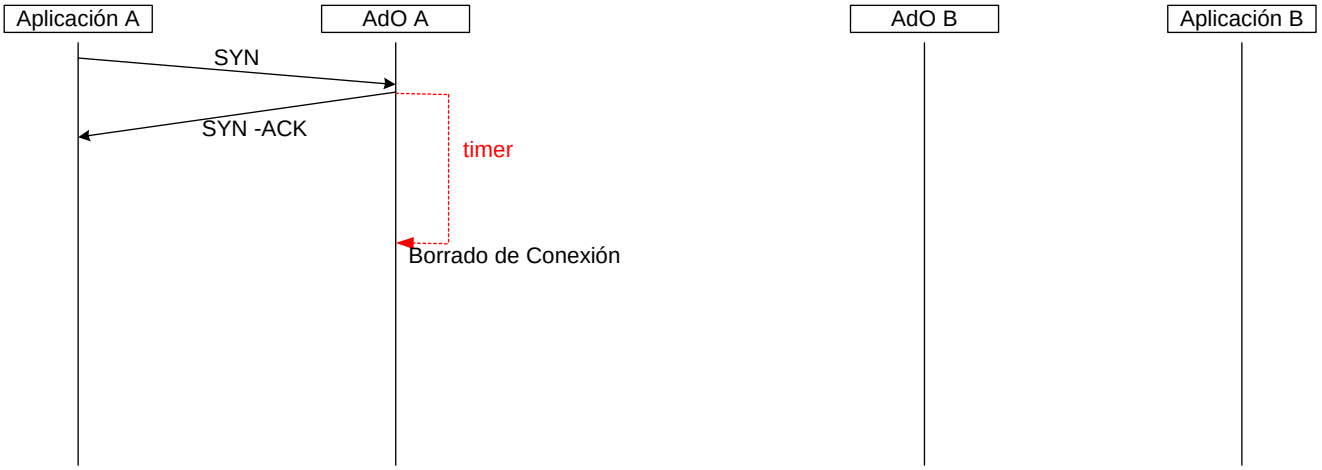


Figura 9 Establecimiento de conexión: Vencimiento de timer

Si recibe un SYN para una conexión ya activa, ignora dicho paquete.

En el establecimiento de la conexión el AdO no negocia ninguna de las opciones de la conexión, por lo que se establecen los valores por defecto.

4.4.2. Envío de Datos Aplicación - AdO

Una vez que la conexión esta establecida, la aplicación puede empezar a enviar datos. Cuando el AdO recibe del Capturador un paquete de datos para una conexión activa, comprueba que es correcto (que el número de secuencia es correcto), lo almacena en un buffer y envía al Capturador un ACK suplantando al destino.

Figura 10: Envío de datos Aplicación – AdO

Si el paquete no es el esperado según el número de secuencia, envía al Capturador 3 ACK's para informar a la aplicación de que no era correcto. No se almacenan paquetes no correctos, ni desordenados. Si se recibe un paquete duplicado (con igual numero de secuencia que otro almacenado en el buffer) se ignora dicho paquete.

Figura 11: Pérdida de paquetes Aplicación AdO

Cada vez que se recibe y almacena un paquete de datos para una conexión, se lanza un temporizador para esperar el siguiente paquete. Cuando se recibe un nuevo paquete, se mata el temporizador anterior y se lanza uno nuevo. Si el temporizador vence, se considera que la aplicación no va a enviar más paquetes por el momento y se envía el buffer al otro AdO vía Redirector, con el protocolo privado SC-SC (cuyo funcionamiento se explica más adelante).

Si el Discriminador recibe un FIN, también se envía el buffer al Redirector sin esperar a que venza el temporizador. El cierre de conexiones se detalla más adelante.

4.4.3. Envío de datos AdO - AdO

Este apartado se va a analizar desde dos puntos de vista: Envío de AdO – AdO y recepción AdO – AdO. También se analiza el caso de las retransmisiones AdO – AdO.

- Envío

El envío de paquetes entre AdO – AdO se inicia cuando ha transcurrido un cierto tiempo desde la recepción del último paquete de datos por parte de la aplicación, o cuando se ha recibido un FIN de la aplicación.

Cuando se envía el buffer al otro AdO (se envía una ráfaga de paquetes), antes de enviar cada paquete se rellenan dos campos especiales utilizados en la comunicación AdO-AdO: *número Total de paquetes* y *posición Actual del paquete*.

El *número total de paquetes* hace referencia al número de paquetes que se van a enviar en esta ráfaga al otro AdO - los almacenados en el buffer. Este número es igual para todos los paquetes dentro de una misma ráfaga.

La *posición Actual del paquete* hace referencia a la posición relativa de un paquete concreto dentro la ráfaga, es decir, el número de orden de dicho paquete. Este número es diferente para cada paquete dentro de una ráfaga, y varía entre 1 y el número total de paquetes almacenados en el buffer.

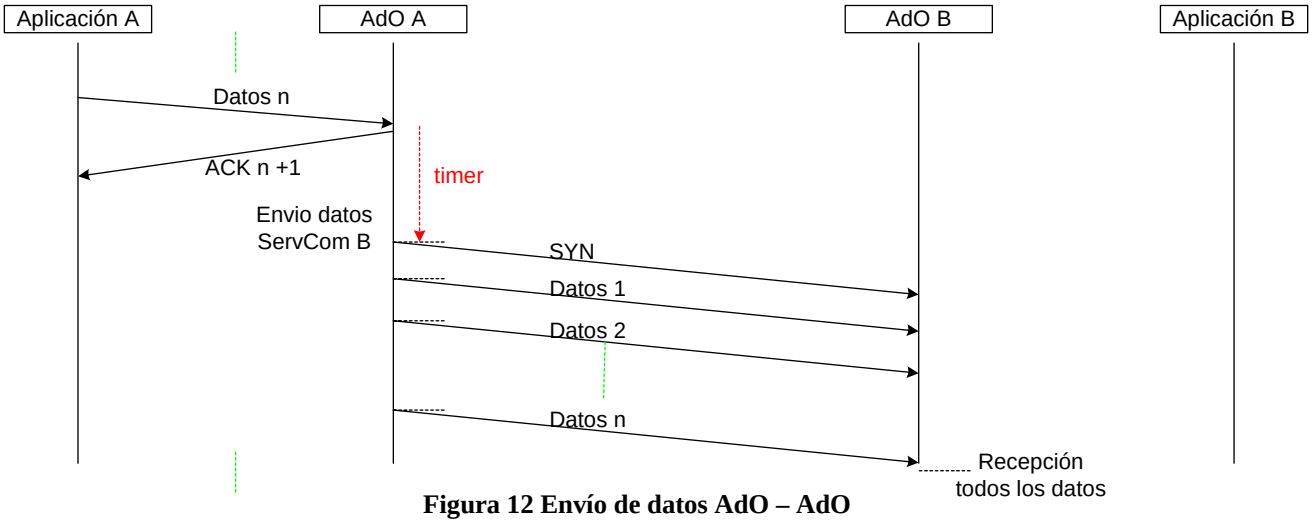


Figura 12 Envío de datos AdO – AdO

Este buffer se almacena hasta que no se reciba confirmación del otro AdO.

Cuando se ha enviado el buffer al Redirector, se lanza un temporizador para esperar la confirmación del otro AdO. Si dicho temporizador vence (no se ha obtenido respuesta del otro AdO), se vuelve a enviar el último paquete almacenado en el buffer con los dos campos especiales rellenados correctamente y se vuelve a lanzar el temporizador. El funcionamiento de los temporizadores se explica más adelante.

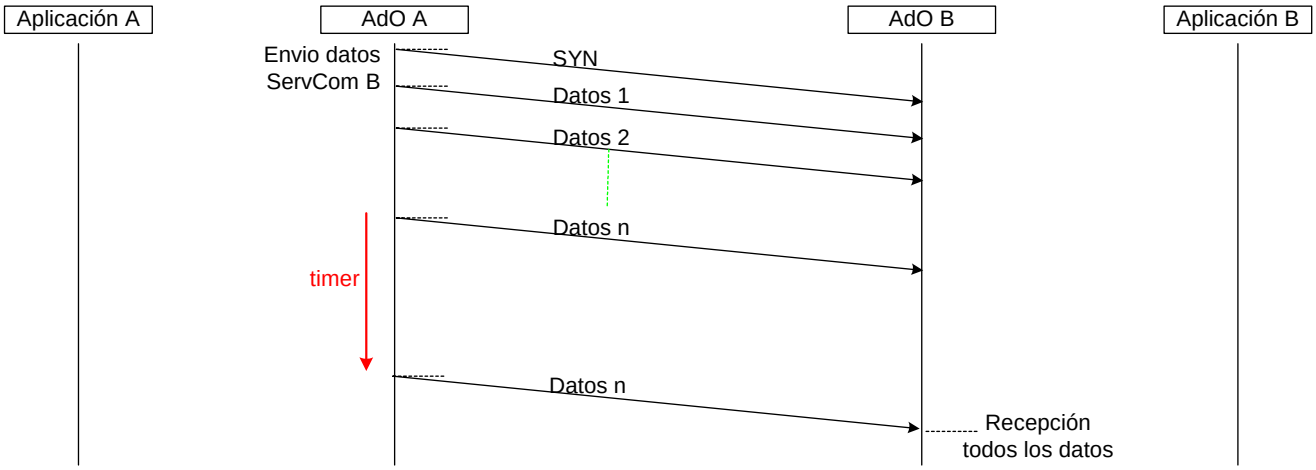


Figura 13: Envío de datos AdO- AdO. Vencimiento del timer

- Recepción

Cuando el AdO recibe un paquete del Adaptador, comprueba los dos campos especiales, si se trata de un paquete de datos, comprueba la conexión, si no existe ninguna conexión para dicho paquete, semiactiva una y almacena el paquete en un buffer. Si ya existía, almacena el paquete de forma ordenada en el buffer, de acuerdo al campo de posición actual del paquete.

Cuando ya ha recibido todos los paquetes indicados en el campo número Total de paquetes, envía el buffer al Capturador, como se explica en los puntos siguientes.

Cada vez que se recibe un paquete del Adaptador para una conexión se lanza un temporizador para la espera del siguiente. Cuando se recibe otro paquete, se mata el temporizador anterior y se lanza uno nuevo, (a no ser que ya se hayan recibido todos los paquetes). Si el temporizador vence, suponemos que se han perdido paquetes, y pedimos retransmisión únicamente de los paquetes que faltan (Retransmisión selectiva).

Como en el envío de paquetes AdO - AdO se habían rellenado dos campos especiales, se sabe en todo momento cuántos paquetes se tienen que recibir del otro AdO, y qué paquetes no hemos recibido aún, de acuerdo con el valor de la posición actual del paquete. Por esto no hay problema en recibir los paquetes desordenados. Los paquetes duplicados se ignoran.

- Retransmisiones

La petición de Retransmisiones AdO – AdO forma parte del protocolo privado. Cuando un AdO quiere pedirle a otro que le envíe un paquete determinado (NACK AdO-AdO), le envía un paquete TCP sin datos, con los campos especiales de la siguiente forma:

- Número Total de paquetes: con un valor de cero (0).
- Posición Actual del paquete: con el valor de la posición relativa del paquete que falta.

Por cada paquete que quiera que el otro AdO le envíe es necesario que envíe un NACK con el número relativo del paquete.

Cuando el AdO recibe del Adaptador un paquete con el campo número Total de paquetes a cero, comprueba que el valor de la posición Actual del paquete se corresponda con un paquete almacenado en el buffer, y envía dicho paquete al AdO que lo ha solicitado.

Si se recibe un NACK se considera como una respuesta y se mata el temporizador para la espera de confirmación del otro AdO y se vuelve a lanzar.

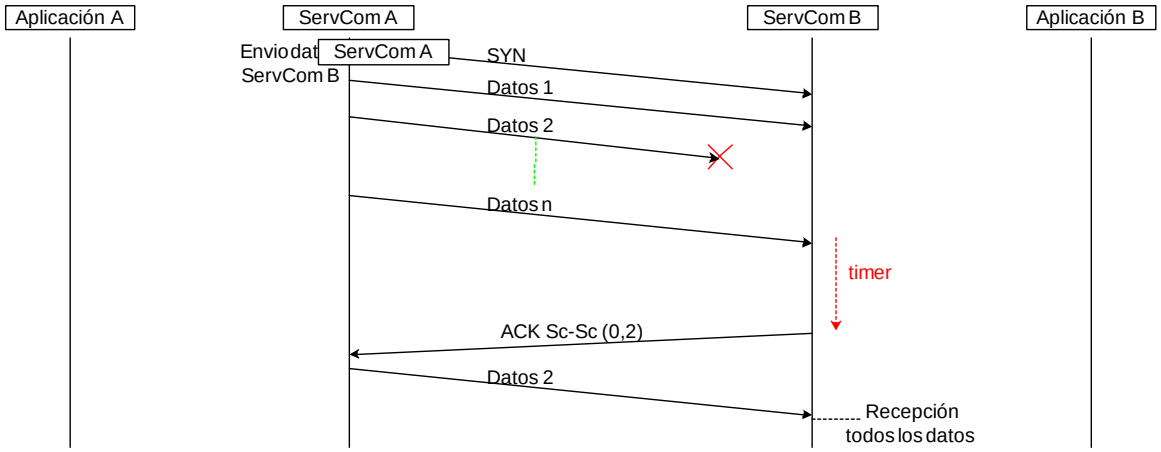


Figura 14 Envío de datos AdO – AdO. Retransmisión

4.4.4. Establecimiento de conexión AdO - Aplicación

Cuando el AdO ha recibido todos los paquetes esperados del Adaptador, se comprueba si la conexión está activa (establecida con la aplicación destino) o semiactiva (establecida con el AdO origen pero no con la aplicación destino).

Si está semiactiva, se envía el SYN al Capturador (el primer paquete del buffer), se lanza un temporizador a la espera de contestación: que el Capturador envíe un SYN-ACK, para poder coger su número de Secuencia inicial. Este número de secuencia se almacena en la información de la conexión, se envía un ACK para terminar el establecimiento y se activa la conexión.

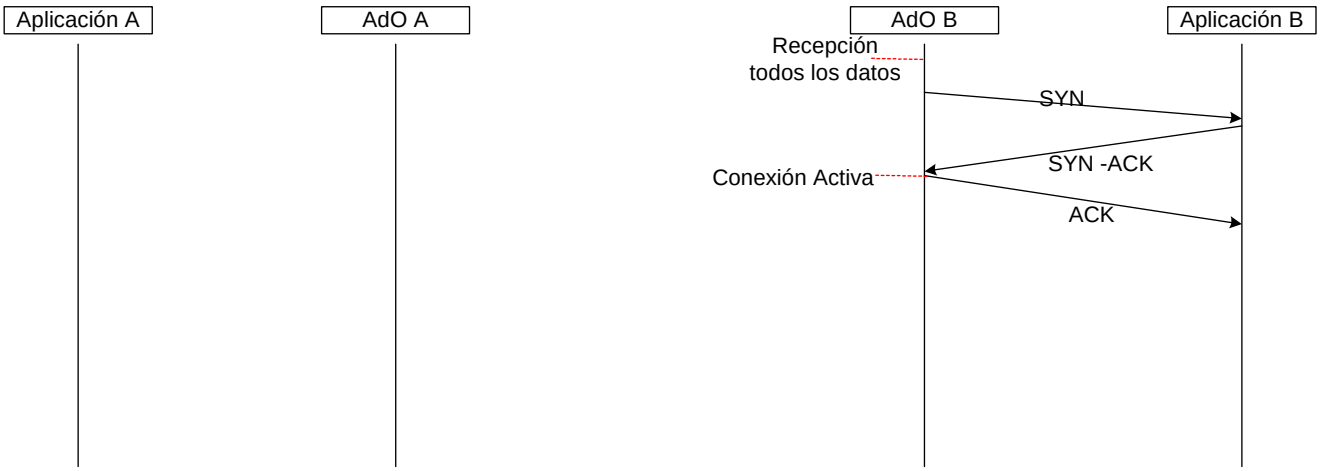


Figura 15 Establecimiento de Conexión AdO - Aplicación

Si vence el temporizador se vuelve a enviar el SYN al Capturador y se lanza el temporizador.

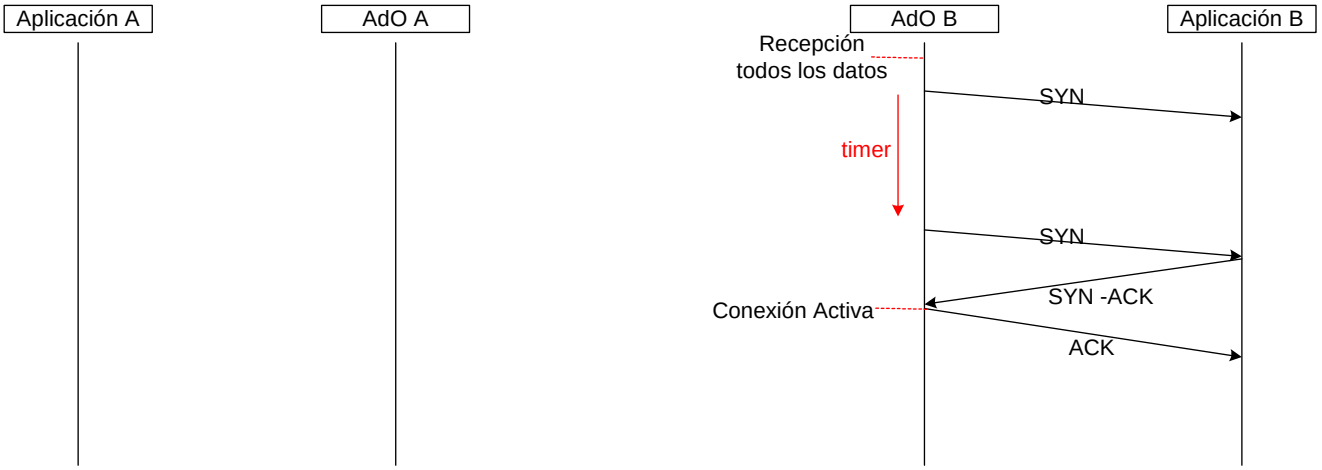


Figura 16 Establecimiento de Conexión AdO – Aplicación. Retransmisión

Tras un determinado número de reintentos, se aborta la conexión, enviándole un reset al AdO remoto para que se lo comunique al origen y libere la conexión.

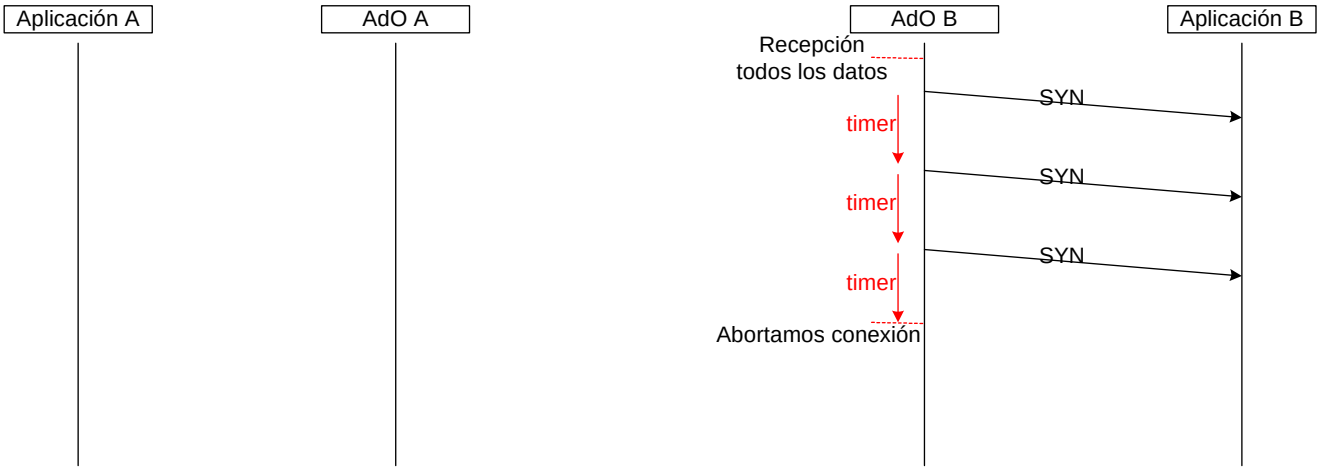


Figura 17 Establecimiento de Conexión AdO – Aplicación

Si se ha completado el establecimiento, se envían los paquetes del buffer a la aplicación.

4.4.5. Envío de datos AdO - Aplicación

Una vez que se ha establecido una conexión TCP entre el AdO y la aplicación, puede empezar el envío de datos.

Para que el envío de datos sea correcto, se debe modificar en todos los paquetes el número de asentimiento, ya que el que poseen hace referencia a la conexión simulada en el otro AdO (origen), y en este extremo se debe usar el proporcionado por la aplicación destino al establecer la conexión (que irá variando de acuerdo con las especificaciones descritas en la RFC 793).

El AdO tiene que enviar el buffer al Capturador, para intentar evitar que la aplicación destino reciba paquetes a una tasa mayor de la que es capaz de procesar, y empiece a disminuir o cerrar su ventana de recepción, se tiene en cuenta el valor de dicho parámetro (ventana de recepción) y se envía el buffer por ráfagas: la primera ráfaga es de tamaño máximo (máximo de paquetes que caben en la ventana de recepción) y el resto de tamaño mitad (mitad de paquetes que caben en la ventana de recepción). Entre ráfaga y ráfaga se espera un tiempo (variable, en función de los paquetes que contenga la ráfaga).

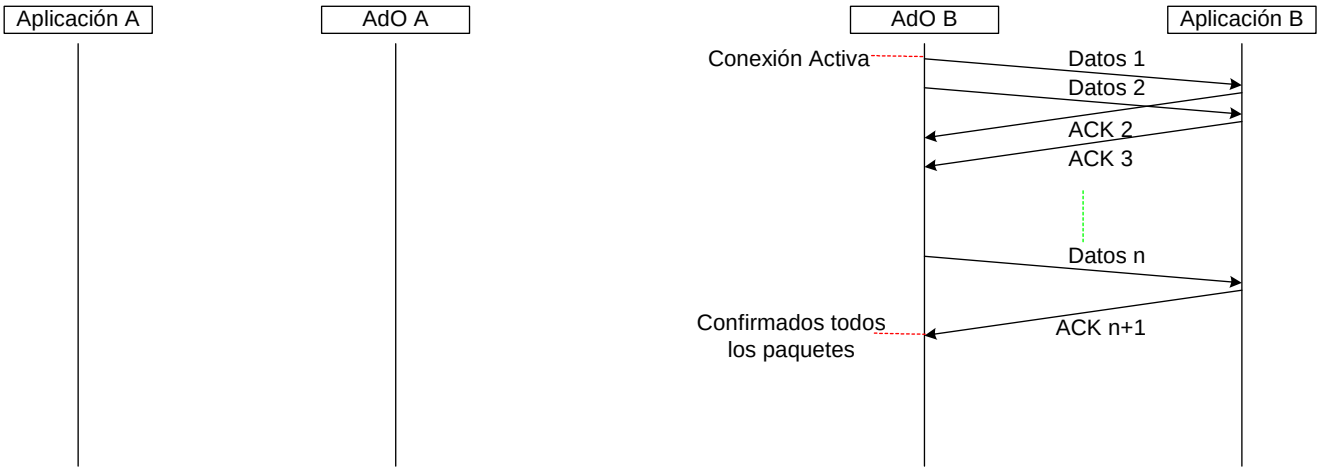


Figura 18: Envío de datos AdO – Aplicación

De manera simultánea, cuando recibimos la confirmación de un paquete de datos (recibimos un ACK por parte de la aplicación), se elimina del buffer. Si la aplicación informa de la pérdida de un paquete, se vuelve a enviar dicho paquete.

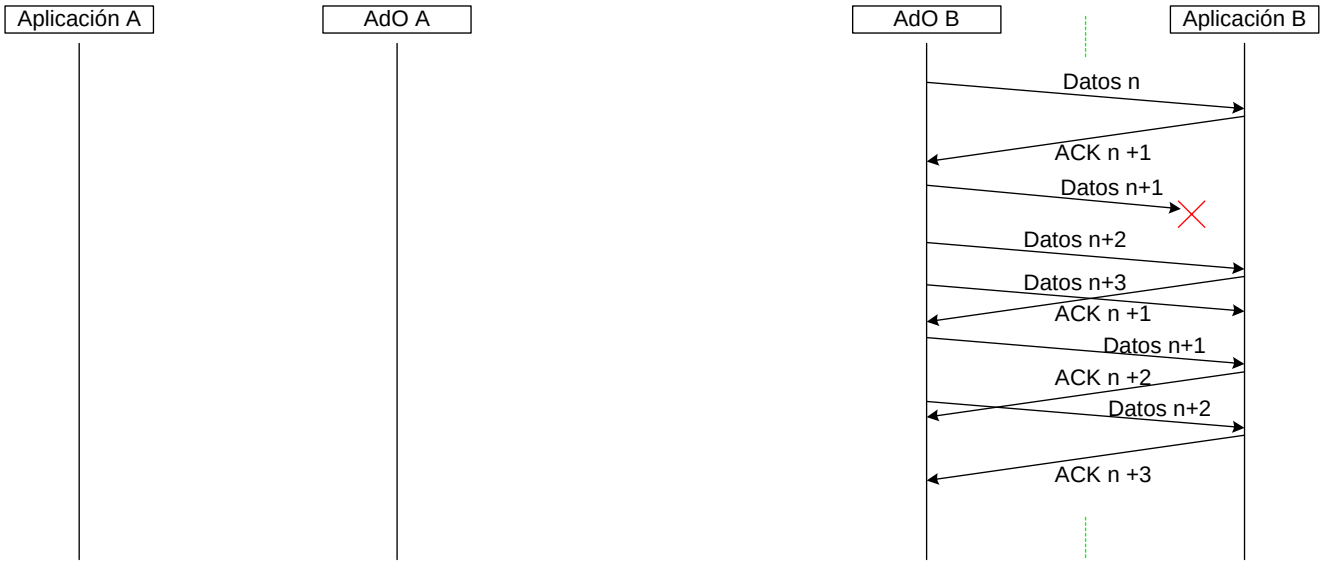


Figura 19 Envío de datos AdO – Aplicación. Pérdida de paquetes

Cuando la aplicación nos haya asentido todos los paquetes del buffer, se envía una confirmación al otro AdO, mediante el protocolo privado de comunicación AdO-AdO. Esta confirmación (ACK AdO-AdO) consiste en un paquete TCP sin datos, con los campos especiales de la siguiente forma:

- Numero Total de paquetes: con un valor de cero (0).
- Posición Actual del paquete: número Total de paquetes recibidos del otro AdO (en esta ráfaga concreta) más uno. Tiene este valor para que el AdO origen no confunda este paquete con un paquete solicitando una retransmisión.

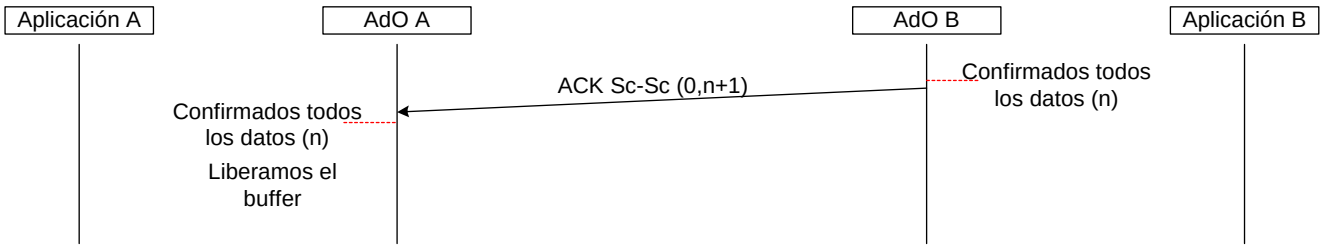


Figura 20 Confirmación de datos AdO – AdO

Cuando el AdO recibe del Adaptador un paquete con estos valores, comprueba su buffer, si el número de paquetes que tiene almacenado es menor que el indicado, significa que la aplicación del otro extremo ya los ha recibido y no va a pedir retransmisiones (ni la aplicación destino ni el AdO), por lo que podemos eliminar de la memoria dichos paquetes. Si el número de paquetes almacenados fuera mayor o igual, se trataría de un NACK AdO-AdO, ya explicado en el apartado anterior.

4.4.6. Cierre de conexión

Cada AdO posee dos listas de conexiones activas:

- Una hace referencia a las conexiones Aplicación – AdO (Lista Capturador)
- La segunda hace referencia a las conexiones AdO – AdO (Lista Adaptador).

Para el cierre de conexiones se consideran 3 casos para cada lista de conexiones activas en ambos extremos de la conexión:

- paquetes recibidos de la aplicación (Lista Capturador)
  + La aplicación NO ha enviado el Fin (caso 0.cap)
  + La aplicación SI ha enviado el Fin (caso 1.cap).
  + El AdO le ha confirmado el Fin a la aplicación (caso 2.cap).
- paquetes recibidos del AdO remoto (Lista Adaptador)
  + El AdO remoto NO ha enviado el Fin (caso 0.adap)
  + El AdO remoto SI ha enviado el Fin (caso 1.adap).
  + El AdO le ha confirmado el Fin al AdO remoto (caso 2.adap).

Inicialmente, cuando se activa una conexión se encuentra en el caso 0.cap y 0.adap.

AdO A		**AdO B**	
Lista Capturador	0	**Lista Capturador**	0
Lista Adaptador	0	**Lista Adaptador**	0

Cuando una aplicación que tiene una conexión activa inicia el cierre de conexiones, envía un paquete con el Flag de FIN activo. El AdO comprueba en que caso se encuentra (0.cap, 1.cap o 2.cap):

- Si esta en el caso 1.cap o 2.cap ignora dicho paquete, el cierre de conexión ya estaba iniciado.
- Si está en el caso 0.cap almacena dicho paquete, cambia al caso 1.cap (AdO origen) y envía el buffer al otro AdO (si hubiera paquetes almacenados se enviarían todos los paquetes, siendo el FIN el último paquete del buffer, y si no, se enviaría únicamente un paquete).

AdO A		**AdO B**	
Lista Capturador	**1**	**Lista Capturador**	0
Lista Adaptador	0	**Lista Adaptador**	0

El AdO destino sigue el comportamiento indicado en puntos anteriores: recibe el buffer y lo envía a la aplicación. Al enviar el FIN a la aplicación cambia al caso 1.adap.

AdO A		**AdO B**	
Lista Capturador	1	**Lista Capturador**	0
Lista Adaptador	0	**Lista Adaptador**	**1**

Cuando la aplicación confirme que acepta que el otro extremo cierre la conexión cambiamos al caso 2.a (AdO destino) y se envía al AdO origen un ACK AdO-AdO.

AdO A		AdO B	
Lista Capturador	1	**Lista Capturador**	0
Lista Adaptador	0	**Lista Adaptador**	**2**

En el AdO origen, cuando recibe del Adaptador un ACK AdO-AdO procede a eliminar el buffer, si en el buffer existe un paquete de FIN, cambia al caso 2.cap (AdO origen) y le confirma a la aplicación el cierre de conexiones en ese extremo.

AdO A		AdO B	
Lista Capturador	**2**	**Lista Capturador**	0
Lista Adaptador	0	**Lista Adaptador**	2

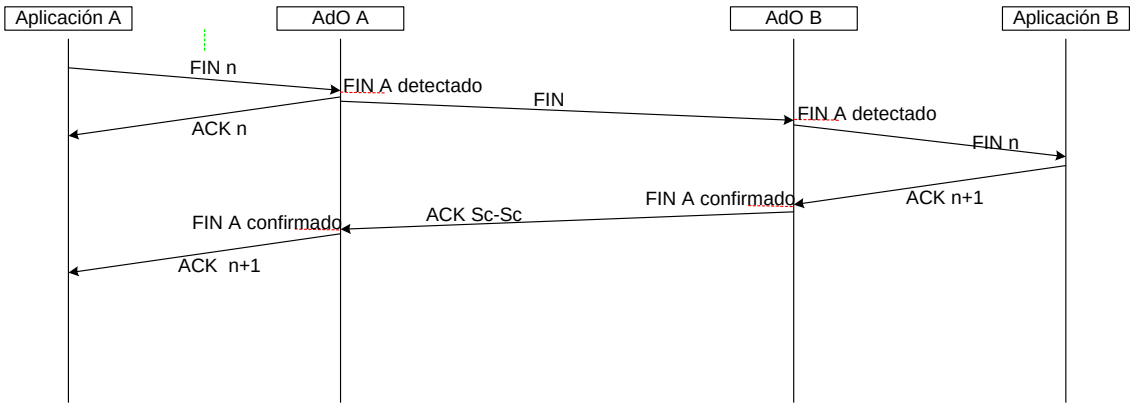


Figura 21 Cierre de conexión 1 extremo

Como se puede ver en la Figura superior, el cierre de la conexión se ha producido de la Aplicación A a la Aplicación B, conviene recordar que según indica la RFC 793, el cierre de conexiones es independiente en cada extremo, un extremo puede cerrar la conexión, el otro confirmársela y seguir enviado datos. La conexión únicamente se cierra cuando ambos extremos han solicitado el cierre y se les ha confirmado. Por tanto, en lo descrito anteriormente únicamente se habría cerrado la conexión en un extremo, pero la conexión seguiría abierta. Es decir, en el ejemplo anterior, el AdO origen se encuentra en los casos 2.cap y 0.adap y el AdO destino en los casos 0.cap y 2.adap.

Cuando la aplicación del AdO B solicite el cierre de conexiones, se procede de la forma ya descrita (el cambio de estados explicados). En cualquiera de los extremos, cuando se permuta al estado 2.cap se comprueba en qué caso se encuentra la lista del Adaptador (y cuando se permuta al estado 2.adap se comprueba en qué caso se encuentra la lista del Capturador). Si un AdO tiene en ambas listas el estado 2, indica que las conexiones se han cerrado en los dos extremos y por tanto se puede liberar la conexión. Tal como indica la RFC 793, la conexión no se elimina inmediatamente si no que se espera un tiempo por si quedasen paquetes aún en la red, aunque dichos paquetes son ignorados. Pasado ese tiempo se elimina la conexión del AdO.

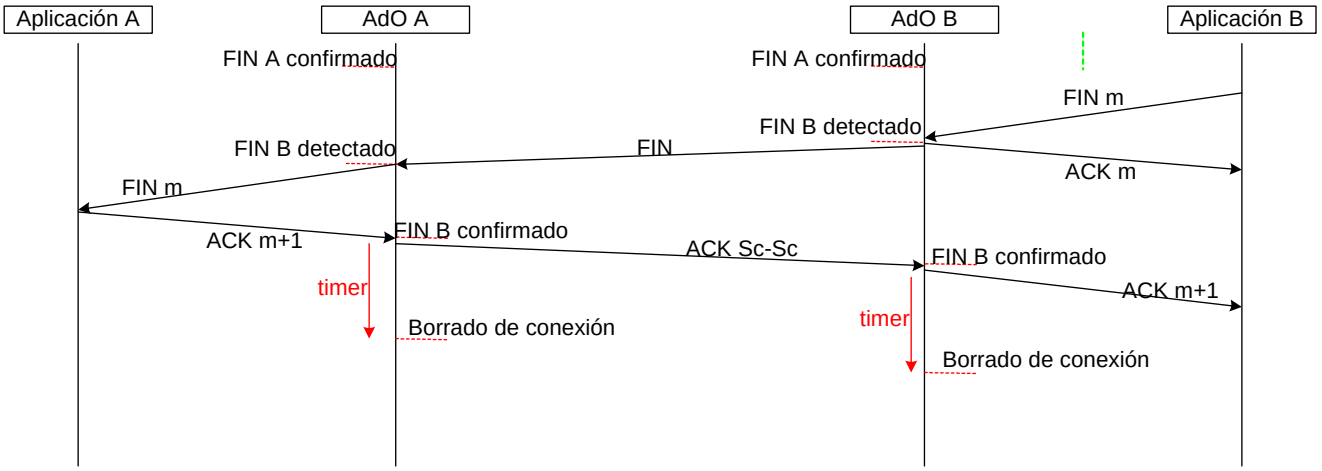


Figura 22 Cierre de conexión 2 extremos

4.5. Casos de interés

En este apartado se explicarán los casos particulares del protocolo TCP que pueden darse en el transcurso de una conexión, como es el caso del envío y recepción de Reset y la limitación de capacidad. También se incluye una descripción de los diferentes temporizadores y un resumen del protocolo privado de comunicación AdO – AdO.

4.5.1. Tratamiento de Reset

El paquete de Reset se usa para abortar una conexión, y se puede enviar por varios motivos: puede haber una pérdida de conexión en un extremo, una falta de sincronización… Para que el paquete sea válido debe tener el número de secuencia adecuado.

Se consideran dos casos: Cuando la aplicación envía un Reset y cuando el AdO envía un Reset (RST).

Si el AdO recibe un RST del Capturador para una conexión que no existe, lo ignora, si la conexión existe y está activa, se activa un flag y se envía un RST al otro AdO. Mientras ese flag este activo todos los paquetes recibidos del Capturador para esa conexión serán ignorados.

Cuando el AdO destino recibe un RST del Adaptador, envía una confirmación al AdO origen, busca si existe una conexión que se corresponda con el RST, y si es así envía un RST a la aplicación y elimina la conexión. Si no existe no se hace nada.

La confirmación de que el AdO destino ha recibido el RST se realiza mediante el protocolo privado SC-SC. Esta confirmación (ACK-RST AdO-AdO) consiste en un paquete TCP sin datos, con los campos especiales de la siguiente forma:

- Número Total de paquetes: con un valor de uno (1).
- Posición Actual del paquete: con un valor de cero (0).

Cuando el AdO recibe del Adaptador un ACK-RST AdO-AdO busca la conexión y la elimina. Si la conexión no existe no hace nada.

Si no recibe respuesta dentro de un tiempo fijado, vence un temporizador y se vuelve a enviar el RST. Si supera el número máximo de reintentos, se borra la conexión.

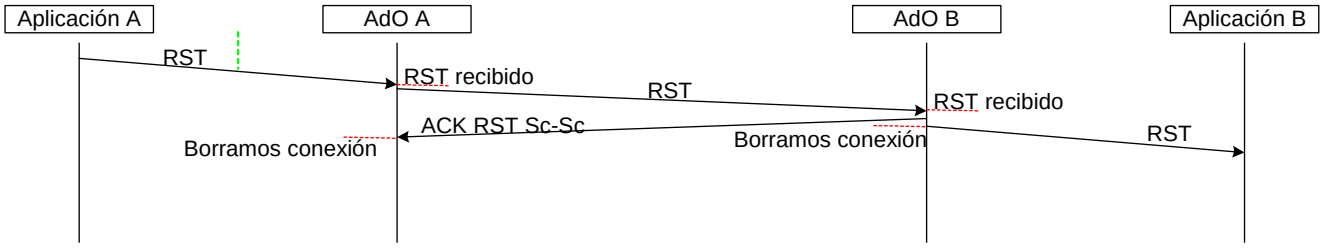


Figura 23 Tratamiento de Reset

Se ha implementado la confirmación del RST AdO-AdO para asegurarnos que ambos extremos se enteran de que existe un problema para esa conexión y debe abortarse, y así evitar que una conexión abortada en un extremo pueda entrar en conflicto si en el otro extremo sigue activa. Por este motivo, la confirmación de RST recibido se envía por triplicado.

Si se recibe un RST para una conexión que está semiactiva, si se trata de un RST a un SYN-ACK, simplemente eliminamos la conexión, y si se trata de un RST a un SYN, enviamos el RST al otro AdO y esperamos confirmación.

El AdO envía un RST a la aplicación en los siguientes casos:

+ Cuando ha recibido RST del AdO remoto.
+ Cuando recibe un paquete de datos, un SYN-ACK o un ACK para una conexión que no se encuentra en su lista de conexiones.
+ Cuando se cierra el Discriminador y hay conexiones activas.
+ Cuando se supera el número de reintentos en la comunicación AdO-AdO.

El AdO envía un RST al AdO remoto en los siguientes casos:

+ Cuando lo recibe de la aplicación para una conexión que existe.
+ Cuando la aplicación no responde al establecimiento de conexión.
+ Cuando se cierra el Discriminador y hay conexiones activas. Cuando cerramos el discriminador no esperamos confirmación del RST

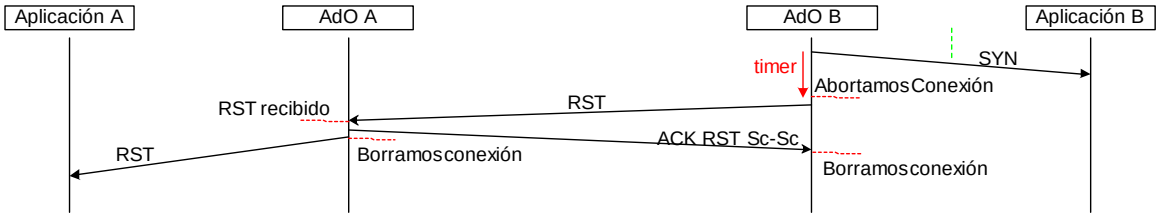


Figura 24 Tratamiento de Reset (2)

4.5.2. Limitación de capacidad

Este algoritmo permite limitar la cantidad de información que se almacena en cada Servidor de Comunicaciones SC para cada conexión. Se desarrolló esta limitación para evitar una ocupación excesiva de las capacidades del sistema, más cuando no es seguro que en el otro extremo la aplicación destino vaya a aceptar la conexión.

El SC posee un interfaz gráfico a través del cual se puede fijar un tamaño de buffer máximo (en Kb). Este tamaño es el máximo que se permite por cada conexión activa, (no se trata de un tamaño máximo que se reparte entre todas las conexiones). Esta limitación sólo la podemos aplicar sobre los paquetes que recibamos del Capturador, para los paquetes que recibamos del Adaptador no se puede establecer ninguna limitación ya que dependen de la configuración que posea el otro AdO, a lo que no se tiene acceso. Únicamente podemos incidir en los datos que envíe una aplicación que pertenezca a la subred del AdO.

El buffer para cada conexión almacena los paquetes recibidos adaptados a la estructura *HeaderPacket*. Por tanto, para saber el número de paquetes que se pueden almacenar habrá que dividir el tamaño máximo configurado entre el tamaño de la estructura *HeaderPacket*. Resaltar que este número de paquetes no se corresponde con el número de paquetes TCP/IP enviados por la aplicación, ya que su tamaño no es el mismo. Por la misma razón, el tamaño del buffer no se corresponde con el tamaño de datos, ya que es necesario almacenar cabeceras y otra información para la gestión de la conexión, que no son datos. Esto se incluye en la estructura *HeaderPacket*.

Cada vez que el AdO recibe un paquete del Capturador, si este es correcto se almacena y se comprueba si el número de paquetes almacenados supera la limitación impuesta, en cuyo caso se activa el flag de congestión, se cierra la ventana de recepción para que la aplicación deje de enviar datos y se envía el buffer al Redirector.

Mientras el flag de congestión este activo, no se almacenará ningún paquete de esa conexión. Si se recibe alguno, se descartará y se notificará a la aplicación que la ventana sigue cerrada.

Cuando el AdO destino nos confirme que su aplicación ha recibido todos los paquetes, el AdO origen liberará su buffer y le notificará a la aplicación que vuelve a abrir su ventana, es decir, que volverá a recibir y almacenar datos de la aplicación.

Este proceso se repetirá tantas veces como fuera necesario durante una conexión.

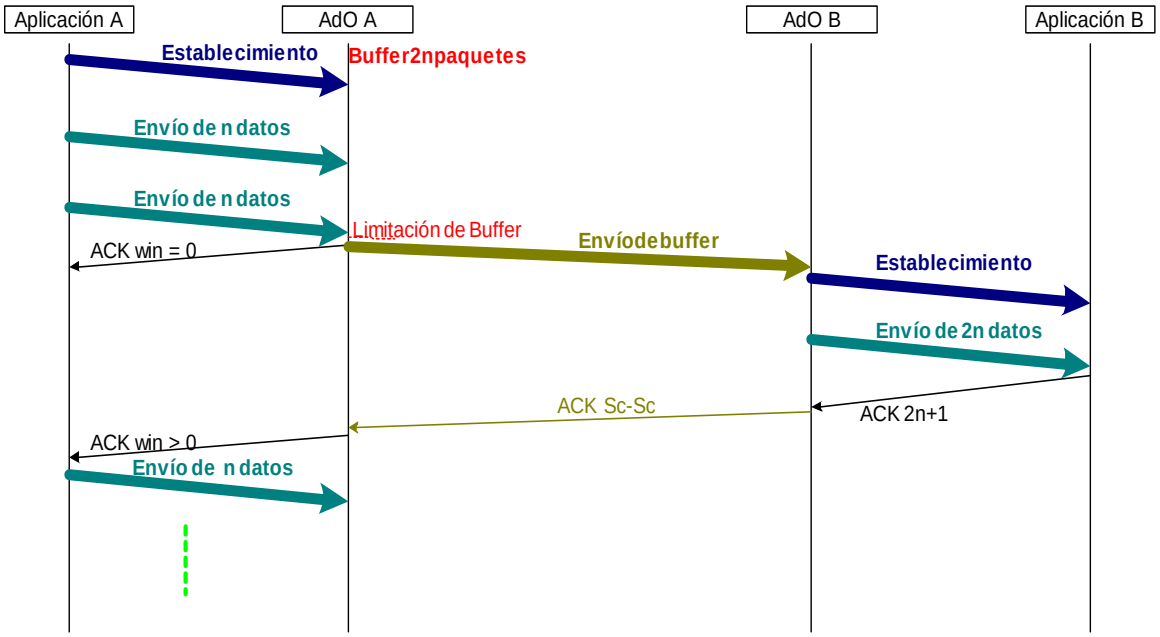


Figura 25 Limitación por capacidad

4.5.3. Configuración

La configuración TCP se realiza a través del interfaz gráfico del Servidor de Comunicaciones, y afecta a los siguientes parámetros:

1-Tamaño máximo del buffer (Kb): indica el tamaño máximo de buffer que el Discriminador puede usar para almacenar paquetes TCP provenientes del Capturador en la suplantación TCP. Cuando ese buffer se llene, se le notificara al Capturador para que no envíe más hasta nuevo aviso, y se enviará dicho buffer al Redirector. Cuando el AdO destino confirme el buffer, se avisará al Capturador que puede reanudar el envío.

El tamaño del buffer es por cada conexión TCP, si hay varias conexiones TCP establecidas, cada una tendrá un buffer de tamaño máximo.

El tamaño máximo solo limita la capacidad para esa conexión en el sentido Capturador -> Discriminador. En ningún caso se limita en el sentido Adaptador -> Discriminador.

2-Número de Retransmisiones: En este documento también se nombra como número de reintentos. Indica el número de veces que intentamos comunicarnos con otro AdO sin recibir respuesta, o que intentamos enviar un SYN sin recibir respuesta. Tras lo cual se aborta la conexión.

3- Tiempo de Tx máximo: Tiempo estimado que tarda un paquete entre SC – SC. Con este valor se configuran los temporizadores que regulan la comunicación AdO – AdO.

4.5.4. Temporizadores e interrupciones

Las rutinas o interrupciones empleadas en los temporizadores son nueve:

1 – Interrupción para el envío de datos al Redirector

Cada vez que se recibe un paquete de datos del Capturador se lanza un temporizador asociado a esta interrupción (y se mata el anterior si hubiera alguno lanzado). Se emplea para detectar si la aplicación ha dejado de enviar paquetes, bien porque no tiene más que enviar, bien porque está esperando respuesta del destino, etc. Si transcurrido un tiempo no se reciben paquetes del Capturador para esa conexión, salta esta interrupción y se envía el buffer al Redirector.

Si el buffer está vacío no se lanza ningún temporizador, solo se lanza esta interrupción cuando existe algún paquete para el otro extremo.

2 – Interrupción para el establecimiento de conexión con la aplicación.

Cuando se recibe un buffer del Adaptador, si la conexión aún no está establecida con la aplicación en este extremo, hay que iniciarla. Para ello se envía un paquete SYN a la aplicación y se lanza un temporizador para asegurarnos que se recibe la contestación (un SYN-ACK). Si transcurrido un tiempo no se ha recibido el SYN-ACK, salta esta interrupción y vuelve a enviar el SYN, si no se obtiene respuesta se repite el proceso hasta que se llegue al número máximo de reintentos que fija la configuración TCP, en cuyo caso se abortaría la conexión y se notificaría al otro AdO mediante un paquete de RST.

3 – Interrupción para la confirmación del establecimiento de conexión con la aplicación

Cuando se recibe un SYN del Capturador, si es correcto, se almacena, se envía un SYN-ACK y se espera recibir un ACK para completar el establecimiento de la conexión. Si no se recibe esta respuesta, se elimina la conexión.

4 – Interrupción para la liberación de conexiones.

Como indica la RFC 793, las conexiones no se eliminan instantáneamente tras el cierre, si no que se espera un tiempo por si hubiese paquetes perdidos en la red. Cuando se ha cerrado la conexión en los dos extremos se lanza un temporizador para su borrado, cuando vence, salta esta interrupción que se encarga de borrar la conexión y liberar la memoria asociada.

5 – Interrupción para abortar conexiones.

Según el protocolo privado SC-SC, en algunos casos se espera a que el otro AdO confirme que ha recibido el paquete de RST. Si pasado un tiempo no se recibe la respuesta, se vuelve a enviar el RST y a lanzar el temporizador. Si se supera el número de reintentos se borra la conexión.

6 – Interrupción para el envío de datos en la comunicación AdO-AdO

Cuando un AdO envía un buffer de paquetes a otro AdO, lanza un temporizador para asegurarse que recibe respuesta del AdO remoto. Si el temporizador vence, se llama a esta interrupción que vuelve a enviar el último paquete almacenado en el buffer y lanza de nuevo el temporizador. Si se supera el número máximo de reintentos establecido en la configuración, se aborta la conexión y se envía un RST al Capturador.

7 – Interrupción para la recepción de datos en la comunicación AdO-AdO

Cada vez que se recibe un paquete del Adaptador para una conexión se lanza un temporizador para comprobar si llegan más paquetes (y se mata el anterior). Si no han llegado todos los paquetes indicados y vence el temporizador, la interrupción comprueba los paquetes que faltan y por cada uno ellos envía una solicitud al AdO remoto para que se los vuelva a enviar, ya que no le han llegado, y vuelve a lanzar el temporizador. Si este proceso se repite sin obtener ningún tipo de respuesta del otro AdO un número mayor que el máximo número de Reintentos permitido, la conexión se aborta y se envía un RST al Capturador.

8 – Interrupción para el cierre de conexiones inactivas

Si una vez establecida la conexión no se produce ningún intercambio de paquetes en un periodo lo suficientemente elevado, se aborta la conexión, si por alguna razón las aplicaciones dejan de responder evitaremos tener almacenada la información eternamente.

9 – Interrupción para la confirmación de datos de la aplicación al AdO

Cuando se envía un buffer de paquetes desde el AdO a la aplicación local, se espera que la aplicación confirme dichos paquetes, si pasado un tiempo no se ha recibido ninguna confirmación se vuelven a enviar los paquetes no confirmados. Tras un número de reintentos se aborta la conexión, enviando un RST al otro extremo.

4.5.5. Comunicación AdO – AdO: Resumen

Para la comunicación AdO-AdO se emplean dos campos: *número Total de paquetes* (usNumPacTotal) y *posición Actual del paquete* (usPacActual). Estos campos hacen referencia a una conexión concreta. En función de los valores de estos campos se tienen diferentes tipos de paquetes:

1- usNumPacTotal > 0 y usPacActual > 0 (usPacActual ≤ usNumPacTotal).

Paquetes de datos para la comunicación AdO-AdO. usNumPacTotal indica el número total de paquetes que se envían / reciben en una ráfaga concreta, y usPacActual la posición del paquete dentro de la ráfaga.

2- usNumPacTotal = 0 y usPacActual > 0.

Paquetes ACK o NACK para la comunicación AdO-AdO. Se emplean para la confirmación de todos los paquetes enviados entre AdO (ACK) o para solicitar retransmisión de un paquete concreto (NACK). Si el valor usPacActual es mayor que el número de paquetes almacenados en el buffer para esa conexión, se trata de un ACK, en caso contrario (NACK) ese valor identifica el paquete a retransmitir.

3- usNumPacTotal = 1 y usPacActual = 0.

Paquete de confirmación de un RST AdO-AdO. Se emplea para confirmar la recepción de un RST del otro AdO. Se utiliza un código especial para evitar posibles confusiones con un ACK AdO-AdO.

El resto de posibilidades no representan ningún tipo de paquete válido para la comunicación AdO-AdO y son ignorados.

Nº Total de Paquetes	**Posición Actual**	**Tipo de paquete**
> 0	>0 y ≤ Nº Total de Paquetes	Datos
= 0	>0 y ≤ Nº Total de Paquetes	NACK
= 0	>Nº Total de Paquetes	ACK
= 1	0	ACK-RST

CAPÍTULO 5:
Descripción del Programa.

5. Memoria Técnica

5.1. Introducción

En el capítulo anterior se ha expuesto el funcionamiento del Algoritmo desde un punto de vista conceptual, intentando dar una visión de qué es lo que hace el Algoritmo. Por el contrario, este capítulo se centra más en la parte práctica, es decir, cómo se ha implementado el comportamiento del Algoritmo. Para ello se describirán las principales estructuras de datos que se han empleado, se detallarán las funciones, se explicarán las pruebas realizadas y se analizarán los resultados.

- El desarrollo de la aplicación se realiza en C/C++ estándar para hacerlo portable a otros sistemas.
- Se utiliza como entorno de desarrollo Borland Builder 6.0

5.2. Estructuras de datos

En este apartado se van a detallar las diferentes estructuras que se han utilizado en el programa, algunas de las cuales ya se han citado en capítulos anteriores, como son:

- La estructura de intercambio para la transferencia de información entre los diferentes módulos.
- Las listas de conexiones presentes en el Algoritmo.

También se describen las variables globales y constantes empleadas.

5.2.1. Estructura de intercambio

La estructura de intercambio se denomina *HeaderPacket*, es la estructura en la que le llegan los paquetes al módulo Discriminador, y por tanto al Algoritmo. También es la estructura utilizada para enviar paquetes a otros módulos desde el Discriminador y el AdO. Contiene información a nivel ethernet, IP y TCP o UDP.

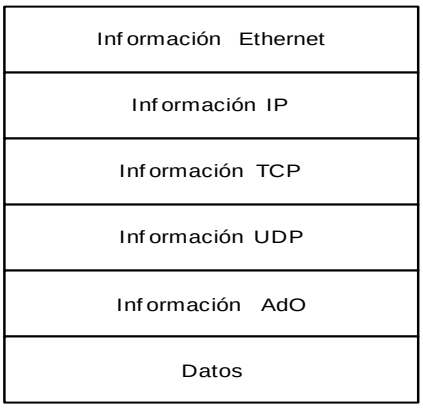


Figura 26 Estructura HeaderPacket

Esta estructura extrae la información de las diferentes capas: Ethernet, IP, TCP o UDP, según corresponda, los datos del paquete, y casi al final, los campos `usNumPacTotal` y `usPacActual` se emplean para la comunicación AdO-AdO, cuyo funcionamiento se ha descrito en el capítulo anterior.

5.2.2. Lista de conexiones

Cada AdO posee dos listas con la información de las conexiones activas que posee. Estas estructuras se dividen en:

- Una lista enlazada con la información del origen de las conexiones. Una entrada por cada origen diferente.
- Una lista enlazada con la información sobre el destino de las conexiones. Una lista por cada origen, y una entrada en la lista por cada destino.
- Una lista enlazada con los paquetes de cada conexión. Una por cada conexión.

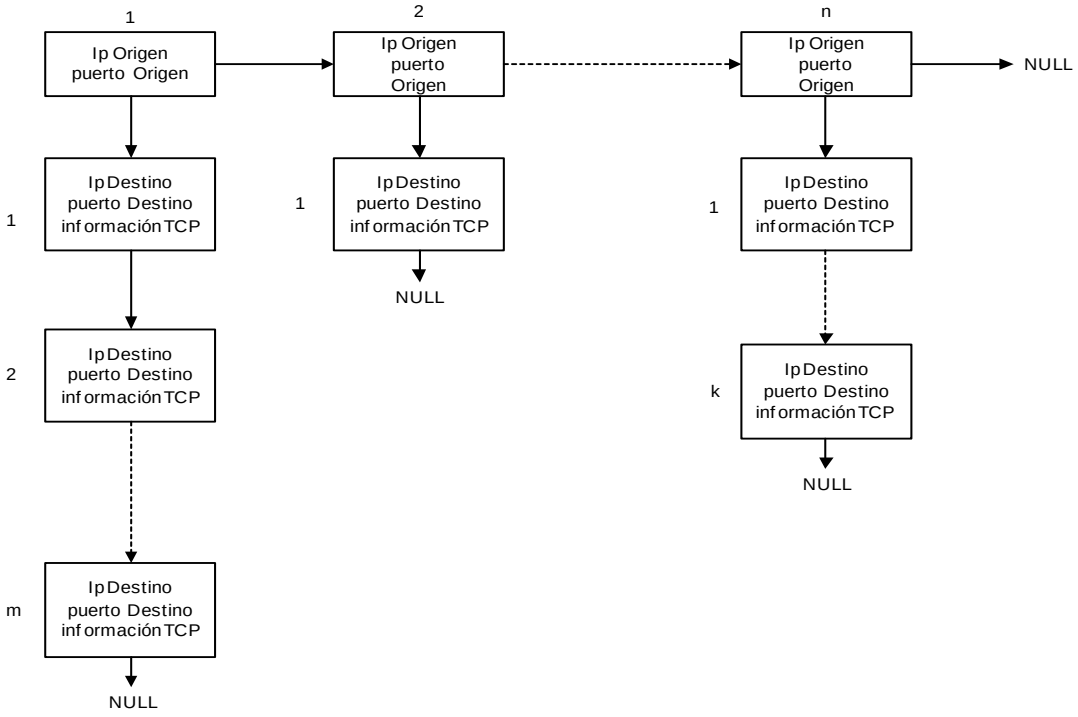


Figura 27 Estructura de Datos

A continuación se describen las diferentes listas:

- **hashTable:**

Es la encargada de almacenar la información del origen de la conexión. Cada entrada en la lista contiene la dirección IP y el puerto origen, un puntero a otra estructura con la información de los destinos y un puntero al siguiente elemento de tipo *hashTable*.

```
typedef struct _hashTable {
   unsigned long  ipSource;
   unsigned short ptoSource;
   _hashTable     *next;
   infoHash       *info;
}hashTable;
```

ipSource: Dirección Ip Origen de una conexión.
ptoSource: Puerto Origen de una conexión.
next: Siguiente elemento hashTable.
info: lista con la información de los destinos activos para este origen.

- **infoHash:**

Esta lista se usa para almacenar la información sobre los destinos para un origen determinado. Cada entrada en la lista contiene información sobre el destino de una conexión, los parámetros necesarios para la optimización TCP, y un puntero a un estructura que almacena los paquetes de la conexión.

```
typedef struct _infoHash {
   unsigned long  ipDest;
   unsigned short ptoDest;
   unsigned long  numAsentimiento;
   unsigned long  numSecPkt;
   unsigned short totalPktCap;
   unsigned short totalPktAda;
   unsigned short rstRecibido;
   unsigned char  endConection;
   unsigned short congestion;
   unsigned short reintentosScSc;
   unsigned long  tamWin;
   HANDLE         hiloRedirector;
   HANDLE         hiloCapturador;
   MMRESULT       timerID;
   MMRESULT       timerScSc;
   bufferPkt      *buffer;
   _infoHash      *next;
   _infoHash      *conexDual;
}infoHash;
```

ipDest: Dirección IP destino de una conexión.
ptoDest: Puerto Destino de una conexión.
numAsentimiento: Número de Secuencia del siguiente paquete que se envíe.
numSecPkt: Número de ACK del siguiente paquete que se envíe.
totalPktCap: Número de paquetes recibidos (o a enviar) del Capturador (Relativo)
totalPktAdap: Número de paquetes recibidos (o a enviar) del Adaptador. (Relativo)
rstRecibido: Indica si se ha recibido un segmento de Reset.
endConection: Indica el estado del cierre de conexiones.
congestion: Indica si se ha superado el tamaño de buffer.
reintentosScSc: Indica el número de veces que se ha retransmitido un paquete
tamWin: Valor de la ventana de recepción de la aplicación.
hiloRedirector: Identificador de la comunicación con el Redirector.
hiloCapturador: Identificador de la comunicación con el Capturador.
timerID: Identificador del temporizador relativo a la comunicación con el Capturador. (Relativo)

timerScSc Identificador del temporizador relativo a las comunicaciones entre ServCom. (Relativo)
buffer: Contiene los paquetes activos en la conexión.
next: Siguiente elemento tipo *infoHash*.
conexDual: Puntero al elemento *infoHash* Dual a este.

- **bufferPkt**

Esta lista ordenada almacena los paquetes activos de una conexión. Se almacena la estructura recibida (*headerPacket*) de la que se ha obtenido la información necesaria. A diferencia de las anteriores, cada entrada tiene un puntero tanto al elemento siguiente como al anterior, ya que los paquetes se almacenan de forma ordenada pero pueden ser recibidos sin orden.

```
typedef struct _bufferPkt{
   struct HeaderPacket pkt;
   _bufferPkt *prev;
   _bufferPkt *next;
}bufferPkt;
```

pkt: Paquete IP (HeaderPacket)
prev: Paquete anterior.
next: Siguiente paquete.

5.2.3. Variables y constantes

Aquí se detallan las principales constantes y variables que se han utilizado:

- MSS
  Tamaño máximo del segmento TCP, por defecto 536 bytes.
- TIME_OVER
  Tiempo de espera entre la recepción de paquetes del Capturador. (milisegundos)
- TIMER_ADAPTADOR
  Tiempo de espera entre la recepción de paquetes del Adaptador. (milisegundos)
- MSL_2
  2*MSL, tiempo de espera desde que se ha cerrado la conexión hasta que se borra. (ms)
- MAX_BUFFER
  Tamaño máximo del buffer de paquetes para una conexión TCP. (nº de paquetes).
- TIME_OVER_SC
  Tiempo de espera en la comunicación AdO-AdO (ms)
- MAX_REINTENTOS;
  Número máximo de veces que se transmite un paquete sin obtener respuesta.
- gHashCapturador
  Variable global con las conexiones activas cuyo origen es el Capturador.
- gHashAdaptador
  Variable global con las conexiones activas cuyo origen es el Adaptador.

5.3. Funciones

En este apartado se van a exponer las principales funciones, ilustrando las explicaciones con diagramas de flujo. Las funciones principales son:

- CheckPktTCPCapturador: Analiza los paquetes que recibe del Capturador.
- CheckPktTCPAdaptador: Analiza los paquetes que recibe del Adaptador.

Estas funciones analizan cada paquete, y en función de qué tipo de paquete TCP es, y del estado de la conexión a la que hace referencia, llama a una función u otra. El resto de funciones son invocadas por estas dos.

5.3.1. CheckPktTCPCapturador

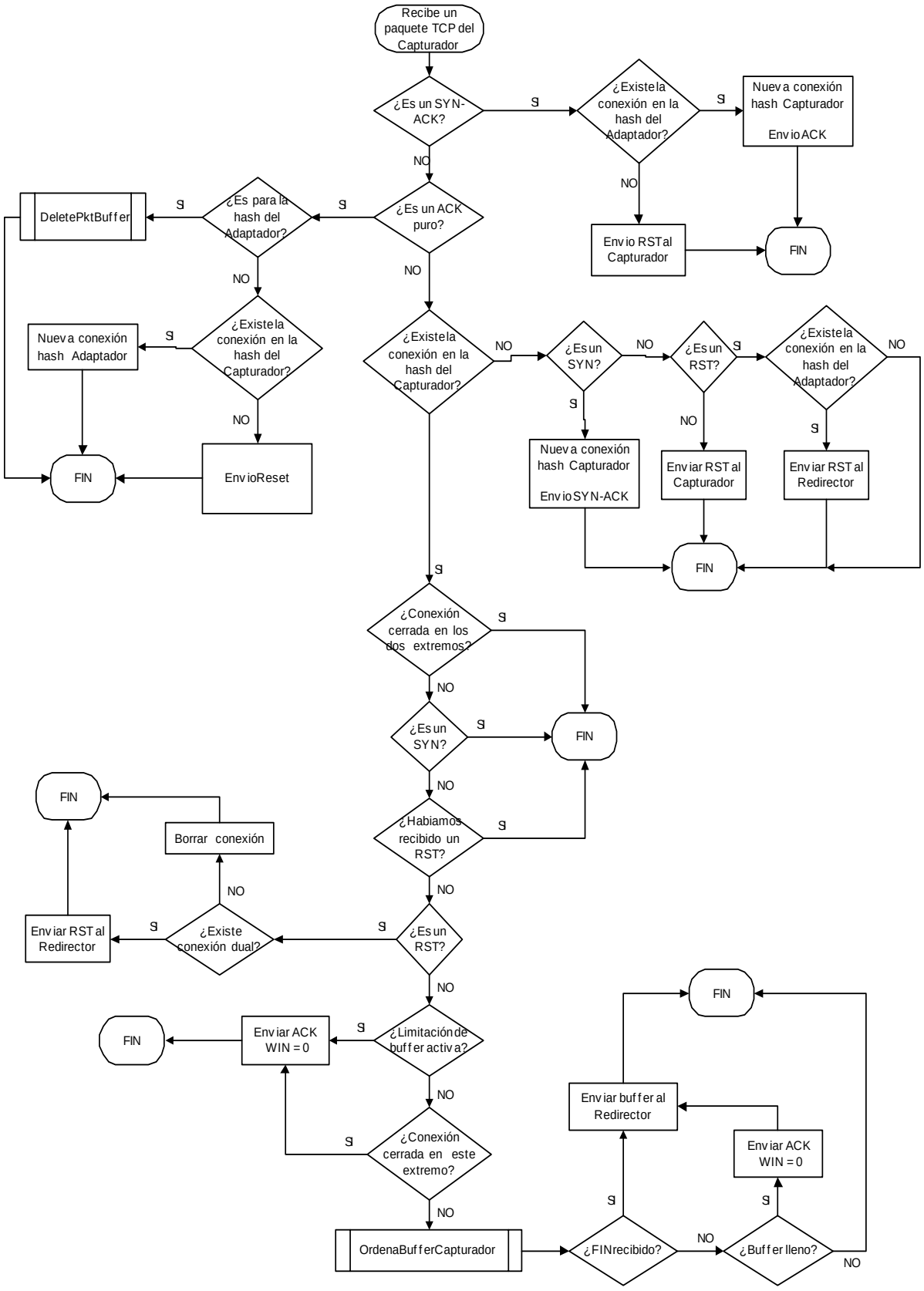


Figura 28 Diagrama de Flujo. CheckPktTCPCapturador

- **Sinopsis**: Recibe un paquete del Capturador y lo analiza en función del protocolo TCP.
- **Parámetros**: *HeaderPacket* * (entrada): Paquete a analizar
- **Descripción**: Comprueba que tipo de paquete ha recibido: SYN, SYN-ACK, FIN, RST, Datos..., si dicho paquete es válido, si pertenece a una conexión ya establecida, y en función de esto le contesta siguiendo los pasos del protocolo TCP establecidos en la RFC 793.

Esta función recibe los paquetes TCP del Capturador, es decir, paquetes TCP de aplicaciones que se encuentren en la misma red local que el Servidor de Comunicaciones. Según el tipo de paquete que sea, se procederá de una forma u otra. Para explicar el funcionamiento de la función se va a exponer el comportamiento para cada tipo de paquete.

Tipos de paquetes que puede recibir del Capturador:

1 – SYN

La aplicación local está iniciando un establecimiento de conexión. El AdO comprueba que no exista ninguna conexión activa en la lista del Capturador (si así fuera se ignoraría dicho paquete), crea una nueva entrada en esa lista (conexión semi-activa, esperando completar el establecimiento) con los datos de ese paquete, envía la respuesta (un SYN-ACK) suplantando al destino y lanza un temporizador para completar el establecimiento. Si vence dicho temporizador sin recibir respuesta, elimina la conexión de la lista.

En caso de que el paquete sea válido, para realizar lo anterior invoca a las funciones:

- *CreateSendSynAck*
- *IntSYNACK*

2 – SYN-ACK

La aplicación local está respondiendo a una petición de establecimiento iniciada por el AdO, el cual debe haber recibido previamente al menos un paquete SYN del AdO remoto a través de los Adaptadores. Debe comprobar que realmente es esa situación, por lo que analiza si existe esa conexión en la lista del Adaptador, si es así, envía un ACK para completar el establecimiento, crea una nueva entrada en la lista del Capturador para esta conexión (conexión activa) y si hubiera más paquetes en el buffer de los Adaptadores, se los enviaría a la aplicación.

Es este caso las funciones involucradas son:

- *getNumSecuence*
- *NewConectionCapturador*
- *CreateSendSynAck*
- *CheckAckAdaptador*
- *SendBuffer2Capturador*

Si no existe esa conexión en la lista de los Adaptadores, o si la conexión ya estuviera establecida, significa que está recibiendo una respuesta para una petición no realizada, por lo que le enviaría un RST a la Aplicación local.

En este caso se llamaría a la función *CreateSendReset.*

3 – ACK

Se diferencian dos casos:

- que sea el tercer paso del establecimiento de conexión.
- que sea un paquete de confirmación de los datos enviados.

Se comprueba si existe una entrada en la lista del Adaptador, si es así se analiza el buffer para ver si se trata de un asentimiento de datos enviados, si este es el caso, se eliminan del buffer los paquetes que haya confirmado, si por el contrario, no confirma datos si no que hace referencia a un paquete perdido, vuelve a enviar los paquetes necesarios. Si el último paquete confirmado es un FIN, actualiza los estados de cierre de conexión y si ambos extremos ya han cerrado, lanza un temporizador para liberar la conexión.

Si no existe la entrada en la lista del Adaptador, comprueba si existe una entrada en la lista del Capturador (si previamente ha recibido un SYN). Si es así, crea una entrada en la lista del Adaptador (conexión activa), y actualiza los datos.

En caso contrario, envía un RST a la aplicación

Dependiendo del caso concreto, pueden intervenir las siguientes funciones:

- *CheckAckAdaptador*
- *searchDirectConnection*
- *NewConectionAdaptador*
- *CreateSendReset*
- *intData*

4 – Datos

Si lo que recibe es un paquete de datos, se comprueba si existe esa conexión en la lista del Capturador, si no es así envía un Reset a la aplicación local que envío el paquete. Si pertenece a una conexión activa se comprueba que el paquete sea correcto, se almacena en el buffer, se envía un ACK confirmando a la aplicación que ese paquete se ha recibido y se lanza un temporizador para esperar más paquetes. Si el paquete no tiene un número de secuencia correcto, envía tres ACK a la aplicación con el número de secuencia esperado y se descarta el paquete.

Si pertenecía a una conexión activa, se analiza también el valor del campo ACK para ver si se puede liberar algún paquete del buffer del Adaptador.

Si el temporizador vence sin haber recibido un nuevo paquete para esa conexión, se envían los paquetes almacenados en el buffer del Capturador al AdO destino mediante el protocolo interno.

Al almacenar un paquete de datos correctos se comprueba que no se haya superado la limitación por capacidad fijada por el usuario, en el caso de que la haya superado, se le envía a la aplicación local un ACK con el tamaño de ventana a cero para evitar que envíe más paquetes de datos hasta que no haya memoria suficiente, y se envía el buffer al otro extremo. Cuando el AdO remoto confirme que ha recibido todos los paquetes, se eliminará en buffer de memoria y se le volverá a abrir la ventana de transmisión mediante un ACK con tamaño de ventana > 0.

Las funciones que son llamadas en este caso son:

- *CreateSendAckWin0*
- *MergeBufferCapturador*
- *SendBuffer2Redirector*
- *int_SC_SC_Capturador*
- *intData*

5 – RST

Si se recibe un RST de la aplicación, se comprueba si existe alguna conexión activa o semiactiva en el AdO. Si no existe, simplemente lo ignora, si sólo está activada en la hash del Capturador se borra dicha conexión de la memoria, pero si está activada en la hash del adaptador, es necesario comunicárselo al AdO remoto, por lo que se le envía un mensaje al otro extremo y cuando se recibe la confirmación se borra la conexión.

Intervienen las siguientes funciones:

- *CreateSendReset*
- *IntRST*
- *DeleteConectionCapturador*
- *SendPkt*

6 – FIN

Cuando se recibe un paquete de fin, se activa el mismo mecanismo que cuando vence el temporizador en la espera de datos, se activa un indicador de que se ha recibido un FIN y se envía todo el buffer al otro extremo con el protocolo interno AdO – AdO, esperando la confirmación de que le han llegado al otro extremo. Cuando el cierre de la conexión se ha solicitado y confirmado por ambos extremos se lanza un temporizador, cuando este vence se elimina la conexión de la memoria. Esto se hace siguiendo las especificaciones de la RFC 793 por si hubiera paquetes “perdidos” para esa conexión.

En este caso se llama a las funciones:

- *SendBuffer2Redirector*
- *int_SC_SC_Capturador*

Dado los diferentes paquetes que se pueden recibir y los diferentes estados de ambas listas, para una mejor compresión se presenta un diagrama de estados:

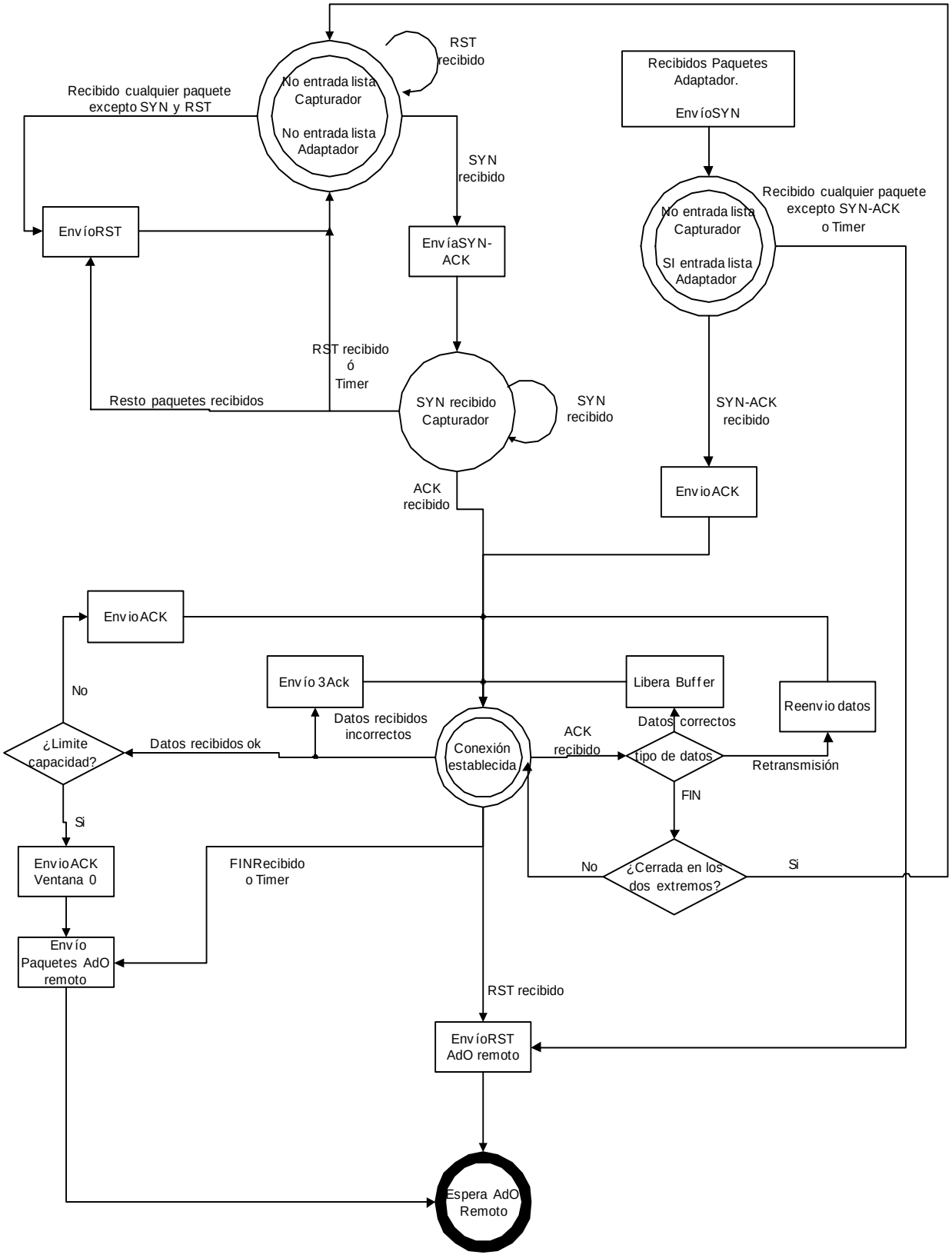


Figura 29 Diagrama de Estados. CheckPktTCPCapturador

No están recogidos todos los estados, ni todos las transacciones posibles, pero sí los más importantes. Recordemos que esta función analiza los paquetes que recibe desde la aplicación local, por lo que podemos tener dos estados iniciales:

- No existe la conexión en ninguna de las dos listas.
- Existe la conexión en la lista del Adaptador, es decir, la aplicación remota es la que ha iniciado la conexión.

En el primer caso, cuando no existe la conexión en ningún extremo, el único paquete válido sería un SYN, se enviaría la contestación SYN-ACK y quedaría a la espera de la confirmación en un nuevo estado intermedio. Un reset sería ignorado, y cualquier otro paquete (FIN, ACK, Datos) es contestado con un RST, permaneciendo en el mismo estado inicial.

En el estado intermedio 'SYN recibido' permanece a la espera del ACK para completar el inicio de la conexión, por lo que si lo recibe pasará al estado 'conexión establecida'. Si recibe un nuevo paquete de SYN es ignorado mientras se esté en este estado. Si recibe un RST o vence el temporizador de inicio de conexión, vuelve al estado inicial (sin conexión en ninguna lista), y si recibe cualquier otro tipo de paquete envía un RST y se sitúa en el estado inicial.

En el segundo estado inicial, está a la espera de que la aplicación local responda a la petición de inicio de conexión, por lo que el único paquete válido sería un SYN-ACK, en cuyo caso envía un ACK y pasaría al estado 'conexión establecida', un paquete de datos, FIN o RST, o el vencimiento del temporizador significaría que se aborta la conexión, por lo que se envía un RST al AdO remoto y se queda a la espera de la confirmación. En el hipotético caso de que se hubiera enviado un SYN a la vez por el AdO y la aplicación, se enviaría un RST y de nuevo el SYN (permaneciendo en el mismo estado).

En el estado principal 'conexión establecida' se espera recibir casi cualquier tipo de paquete.

- Datos Correctos: se almacenan y se comprueba el límite de capacidad, si no lo ha superado se envía un ACK a la aplicación y permanece en el mismo estado, si por el contrario se ha superado, se envía a la aplicación un ACK con la ventana a 0 y se envían al AdO remoto, quedando a la espera de su contestación.
- Datos incorrectos: si los datos no son los esperados se envían 3 ACK's con el número de secuencia esperado para que la aplicación los reenvíe, permaneciendo en el mismo estado.
- Si se recibe un FIN o vence el temporizador, se envía el buffer al AdO remoto y se queda a la espera de la contestación.
- Si se recibe un RST se envía el RST al AdO remoto y espera confirmación.
- Si recibe un ACK, comprueba si es una confirmación a datos enviados desde el AdO a la aplicación, en cuyo caso se libera el buffer y vuelve al mismo estado, si solicita retransmisión de paquetes, en cuyo caso se envían de nuevo, permaneciendo en el mismo estado, o si es un ACK a un FIN, si la conexión esta cerrada en ambos extremos la libera, pasando al estado inicial ('Sin conexión'), y si sólo está en el extremo remoto, le envía la confirmación al AdO remoto y permanece en el mismo estado.

Mientras está a la espera de confirmación del otro AdO, puede suceder que se reciban paquetes, dependiendo del evento que provocara el cambio de estado actuará de una forma u otra:

- Si se recibió un RST o se envío uno al otro extremo, mientras no lo confirme el AdO remoto cualquier tipo de paquete será ignorado.
- Si está activa la limitación por capacidad, a la espera de confirmación de FIN o venció el temporizador, se contesta con un ACK con la ventana a 0, para indicar que la conexión sigue viva, pero no acepta los datos nuevos. Si se recibe un RST se enviaría al otro extremo y esperaría la confirmación del RST para eliminar la confirmación.

5.3.2. CheckPktTCPAdaptador

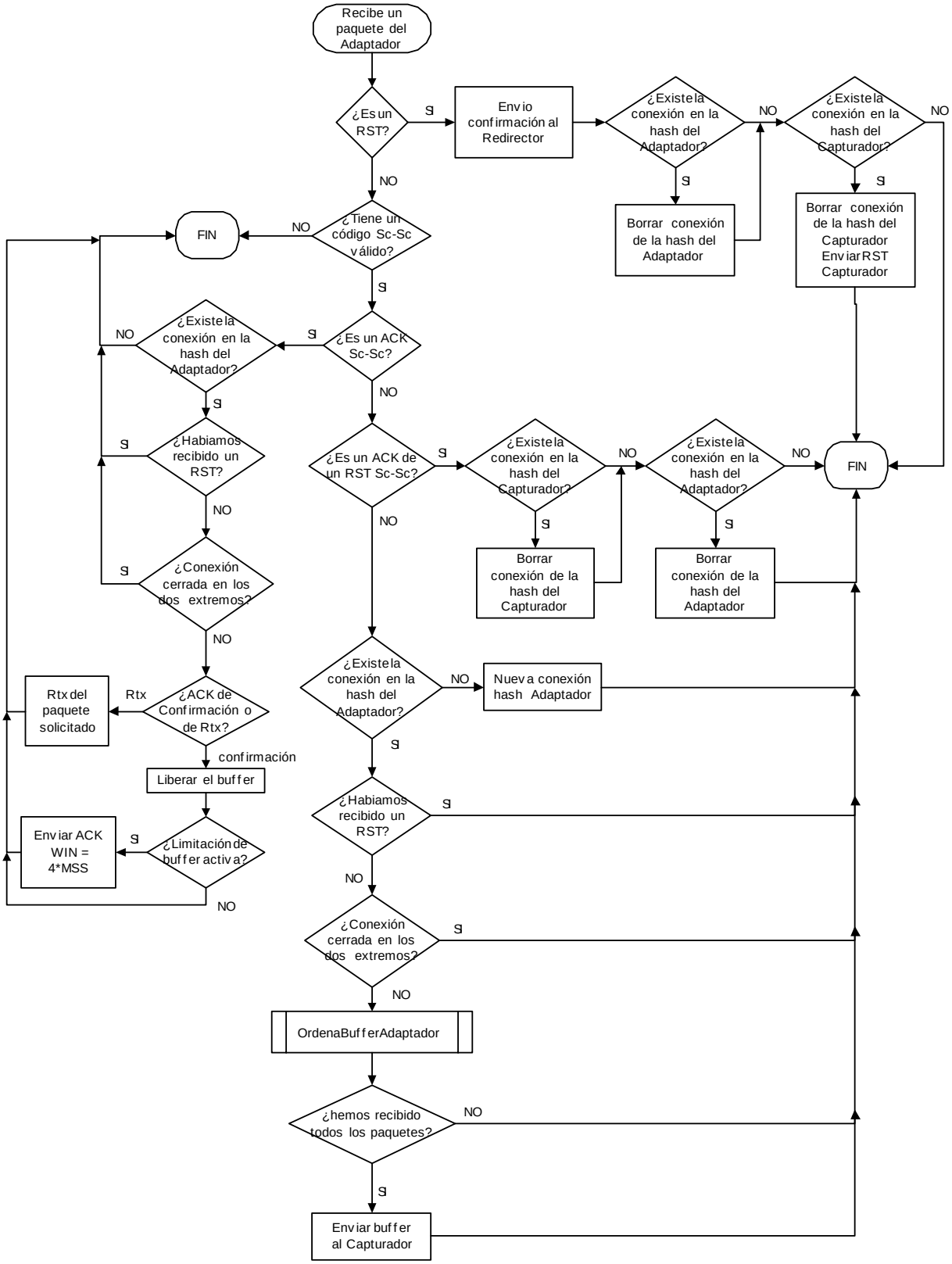


Figura 30 Diagrama de Flujo. CheckPktTCPAdaptador

- **Sinopsis**: Recibe un paquete del Adaptador, lo analiza en función de la optimización del protocolo TCP.
- **Parámetros**: Fifo_Adapter *(entrada): el paquete y la velocidad del canal
- **Descripción**: Comprueba que tipo de paquete ha recibido: RST, ACK, Datos..., si dicho paquete es valido, si pertenece a una conexión ya establecida, y en función de esto le contesta siguiendo los pasos del protocolo TCP establecidos en la RFC 793

Esta función recibe los paquetes TCP de los Adaptadores, es decir, paquetes TCP del AdO remoto. Según el tipo de paquete que sea, se procederá de una forma u otra. Para explicar el funcionamiento de la función se va a exponer el comportamiento para cada tipo de paquete.

Tipos de paquetes que podemos recibir del Adaptador:

1 – RESET

Si el paquete recibido tiene activo el flag de reset, se envía la confirmación de Reset recibido al otro extremo, para que pueda liberar la conexión. A continuación, busca en la lista del Adaptador si existe una conexión que corresponda a ese paquete, si es así, se matan los temporizadores asociados y se borra la conexión de la lista del Adaptador, si también existe la conexión en la lista del Capturador, se envía un Reset a la aplicación, se matan los temporizadores y se elimina la conexión de la lista del Capturador. Si no existe la conexión en la hash del Adaptador, pero si existe en la hash del Capturador, se le envía un Reset a la aplicación y se elimina la conexión.

Las funciones que intervienen en este caso son:

- *CreateSendAckReset*
- *searchDirectConnection*
- *searchReverseConnection*
- *DeleteConectionAdaptador*
- *DeleteConectionCapturador*
- *CreateSendReset*
- *CreateSendResetDual*

2 – ACK comunicación AdO- AdO

Tras enviar un buffer de paquetes al otro extremo, se espera recibir una confirmación de que se han recibido todos los paquetes, para poder eliminar el buffer. Si el paquete recibido es un ACK AdO-AdO, se busca la conexión a la que referencia, y comprueba si es una confirmación de todos los paquetes o si bien se trata de una petición de retransmisión de un paquete concreto.

Si el otro extremo confirma que todos los paquetes se han recibido correctamente, se elimina el buffer almacenado, si había una solicitud de cierre de conexión en el buffer, se confirma que se ha asentido, y si había congestión debido al tamaño definido por el usuario se vuelve a abrir la ventana de transmisión a la aplicación.

Si el extremo remoto solicita la retransmisión de un paquete, se le reenvía el paquete solicitado, se reinicia el contador de reintentos y el temporizador, y se vuelve a esperar la confirmación del buffer. Si el paquete no hace referencia a ninguna conexión se ignora.

Se invoca a las siguientes funciones:

- *searchReverseConnection*
- *DeleteBuffer*
- *CreateSendAckWin0*
- *RtxPkt*
- *int_SC_SC_Capturador*

3 – ACK reset

Si el código del paquete recibido corresponde a la confirmación de un reset, se busca en ambas listas la conexión correspondiente y se elimina. Para ello utiliza las funciones:

- *searchReverseConnection*
- *searchDirectConnection*
- *DeleteConectionCapturador*
- *DeleteConectionAdaptador*

4 – Paquete de Datos

Cuando se recibe un paquete con el código interno de paquete de datos, se comprueba si existe una entrada en la lista del Adaptador para esa conexión.

Si no existe, se trata del primer paquete que se recibe para esa conexión, por lo que se extraen todos los datos de la conexión y se crea una nueva entrada en la lista. Se extrae también el número total de paquetes que se envían en esa ráfaga. Si es el único paquete de la ráfaga se envía al Capturador, y si hay paquetes de esa ráfaga pendientes de recibir, se lanza un temporizador para esperar la llegada del resto.

Si existe ya una entrada en la lista para esa conexión, se comprueba si previamente se había recibido un Reset o cerrado la conexión, en cuyo caso se ignora dicho paquete, si no es el caso se almacena el .paquete de forma ordenada, y se comprueba si se han recibido todos los paquetes de esa conexión, si aún faltan paquetes se lanza de nuevo el temporizador de espera a recibir todos los paquetes y se queda a la espera. Si ya se han recibido el número total de paquetes se procede a enviar los paquetes al Capturador, diferenciando dos casos: si el primer paquete almacenado en el buffer es un SYN, por lo que únicamente se enviaría un paquete SYN al Capturador y se activaría la rutina de inicio de conexión, y si no lo es se activaría la rutina de envío de datos hacía el Capturador.

Las funciones empleadas son:

- *MergeBufferAdaptador*
- *CreateSendSynAck*

- *IntSYN*
- *SendBuffer2CapturadorDual*
- *int_SC_SC_Adaptador*

X – Código no válido

Si recibe un paquete que no es un Reset ni tiene un código de la comunicación AdO-AdO válido se ignora el paquete.

Como se vio en la función anterior, en el diagrama de estados quedaban algunos casos a la espera recibir una contestación por el AdO remoto, que es precisamente lo que hace esta función.

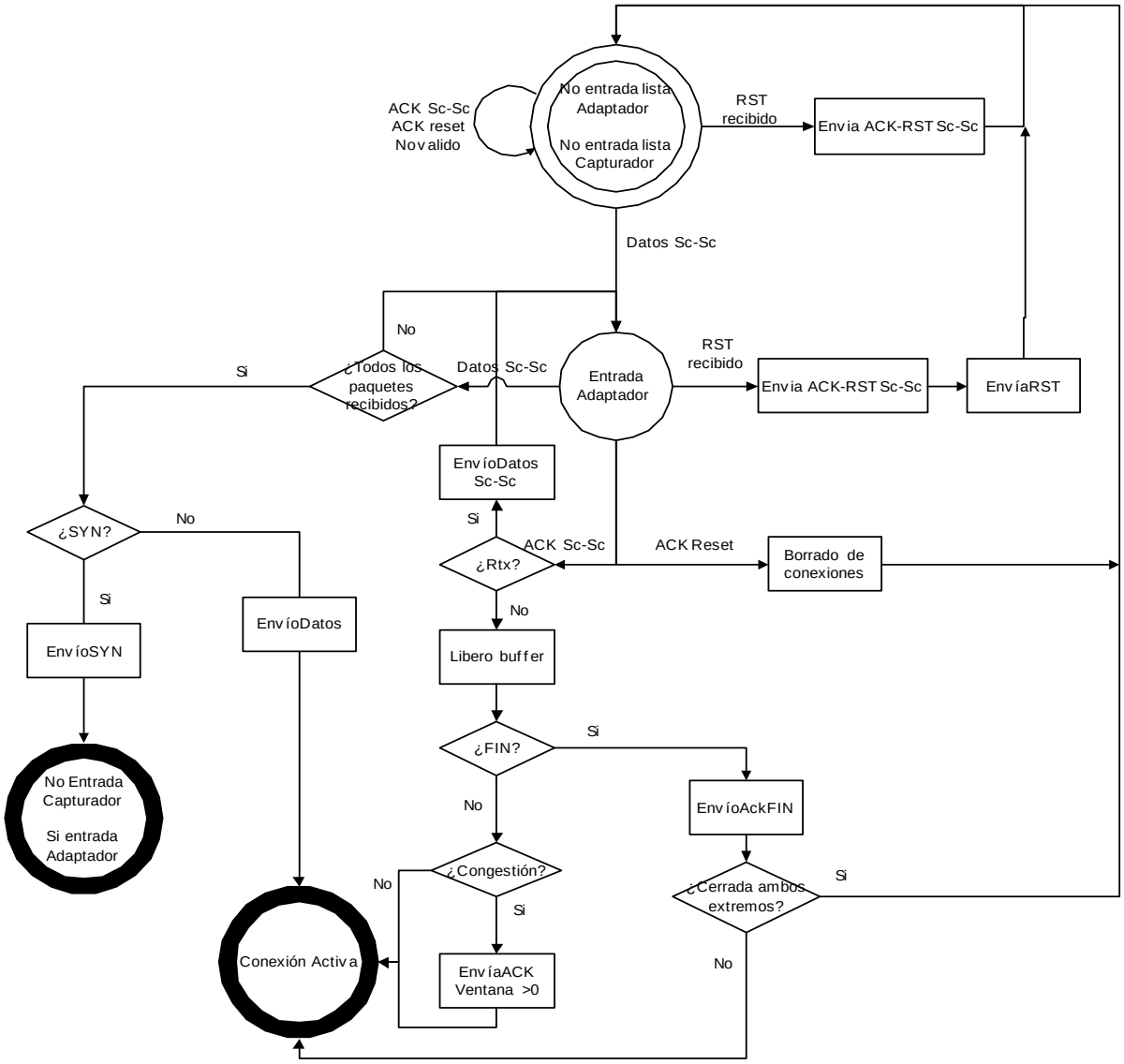


Figura 31 Diagrama de Estado. CheckPktTCPAdaptador

Esta función únicamente recibe paquetes de los Adaptadores, es decir, del AdO remoto. Partimos de un estado inicial en el cual no existe la conexión, por lo que si llega un ACK Sc-Sc o un ACK reset será ignorado, si llega un RST se envía la confirmación al AdO remoto, permaneciendo en el mismo estado, únicamente cambia de estado si le llega un paquete de datos Sc-Sc, cambiando al estado 'entrada Adaptador'.

En este nuevo estado, el comportamiento según el tipo de paquete recibido es el siguiente:

- Datos Sc-Sc: Comprueban si se han recibido todos los paquetes, si falta alguno permanece en su estado, si ya se ha completado el envío comprueba si el primer paquete es un SYN, por lo que hay que establecer la conexión con la aplicación local, le envía un SYN y pasa a uno de los estados iniciales vistos en el apartado anterior, queda a la espera de respuesta por parte del Capturador. Si la conexión ya está establecida envía los datos, pasa al estado 'Conexión activa' también visto en el apartado anterior.

- ACK Sc-Sc: Comprueba si confirma todos los paquetes o solicita retransmisión. Si falta algún paquete se lo envía al AdO remoto y permanece en el mismo estado, y si confirma todos los paquetes libera el buffer de memoria y comprueba si había un cierre de conexión pendiente, si es así envía la confirmación a la Aplicación, si está cerrada en ambos extremos se libera pasando al estado inicial, y si sólo lo está en uno pasaría al estado 'Conexión activa' de la función anterior. Si no hay un cierre de conexión, se comprueba si hay una limitación de capacidad activa, como ya se ha liberado el buffer, si estaba activa se envía a la Aplicación un ACK con un tamaño de ventana mayor que cero para que sepa que puede enviar datos de nuevo, y en ambos casos se pasa al estado 'conexión activa'.

- Si se recibe un ACK reset y efectivamente se había enviado, se liberan las conexiones y se vuelve al estado inicial. Si no se había enviado ningún reset antes se ignora el paquete permaneciendo en el mismo estado.

- RST: si recibe un reset para una conexión activa envía la confirmación de reset al AdO remoto (ACK reset), y libera las conexiones, enviando un RST a la aplicación si fuera necesario, volviendo al estado inicial.

Hay que comentar que varios estados hacen referencia a la lista del Capturador, que como se vio en el apartado anterior puede quedarse a la espera de una respuesta del AdO remoto, dicha respuesta será recibida por los Adaptadores y analizada por esta función, activando un estado u otro de la lista del Capturador.

5.3.3. CheckAckAdaptador

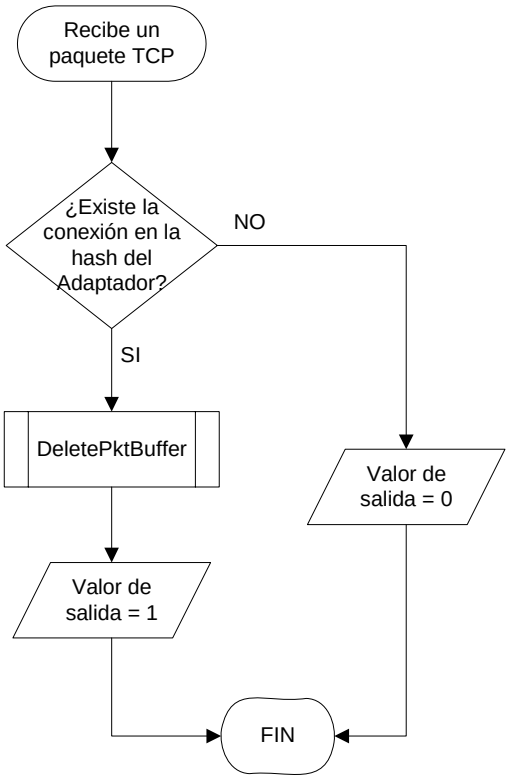


Figura 32 Diagrama de Flujo. CheckAckAdaptador

- **Sinopsis**: Recibe un paquete ACK para el Adaptador, borra del buffer los paquetes confirmados
- **Parámetros**: *HeaderPacket* * (entrada): Ack recibido
  int (salida): 1 si existe la conexión, 0 si no existe
- **Descripción**: La función recibe un paquete TCP que es un ACK, busca en la lista del Adaptador si existe una conexión para la Ip-Puerto origen y destino, y si es así borra los paquetes correspondientes.

Esta función es llamada desde *CheckPktTCPAdaptador* y desde *MergeBufferCapturador*, es decir, siempre que se recibe un paquete del Capturador con el flag ACK activo, por si hubiera paquetes almacenados en la lista del Adaptador que estén a la espera de confirmación. Si existe la conexión en la lista del Adaptador invoca a la función *DeletePktBuffer* para que borre del buffer. El valor de retorno es 1 si existe la conexión en la lista del Adaptador y 0 si no existe

5.3.4. MergeBufferCapturador

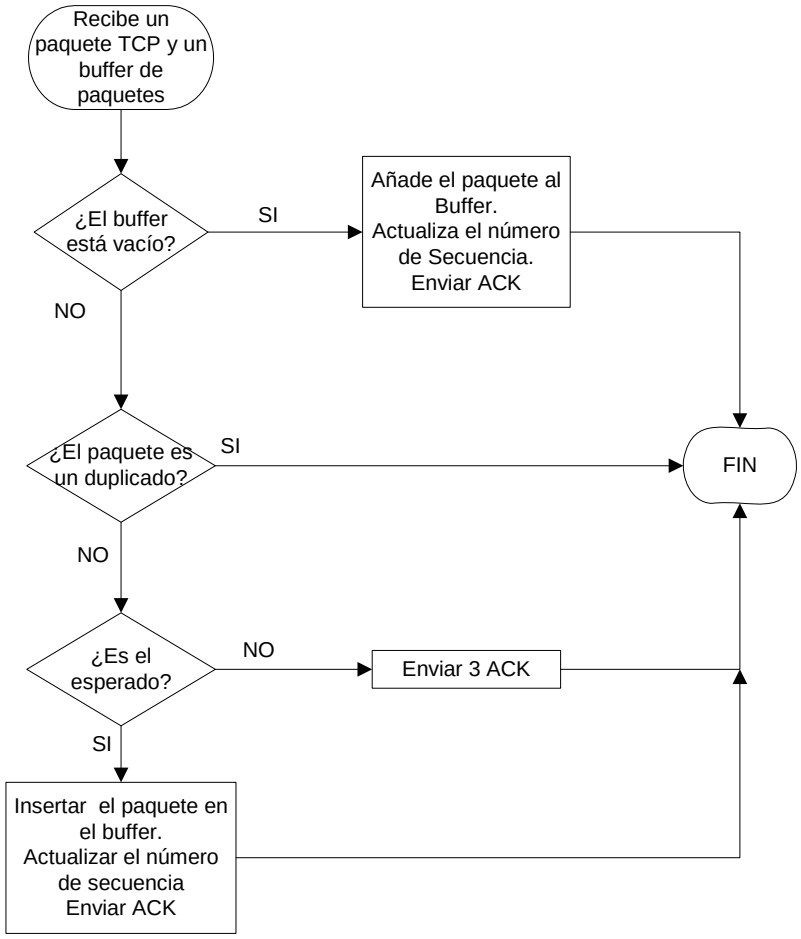


Figura 33 Diagrama de Flujo. MergeBufferCapturador

- **Sinopsis**: Inserta un paquete en el buffer de la conexión.
- **Parámetros**: *HeaderPacket* * (entrada): paquete a insertar
  infoHash * (entrada): Información de la conexión y el buffer de paquetes
  BufferPkt * (salida): puntero al buffer ordenado
- **Descripción**: Comprueba el estado del buffer y almacena el paquete si este es correcto, es decir, si su número de secuencia coincide con el esperado, por tanto siempre se inserta al final del buffer. Devuelve un puntero al inicio del buffer ordenado.

Es invocada desde la función *CheckPktTCPCapturador* cada vez que se recibe un paquete de datos para una conexión activa, y se encarga de actualizar el buffer de paquetes y el número de secuencia.

Si el buffer esta vacío, crea una nueva entrada, actualiza los parámetros de la lista y envía un ACK a la Aplicación confirmando la recepción de los datos.

Si el buffer ya contiene algún paquete, lo recorre, si el número de secuencia del paquete recibido es igual o menor que el almacenado en el buffer, es un paquete duplicado y lo ignora.

Si el número de secuencia del paquete recibido es mayor que el del último paquete almacenado en el buffer, y es el número de secuencia esperado, se añade al final y se envía un ACK a la Aplicación, si no es el esperado, se envían 3 paquetes ACK con el número de secuencia esperado, para que sea retransmitido el paquete deseado.

Invoca a las funciones *CheckAckAdaptador* para comprobar el número de asentimiento del paquete de datos, y *CreateSendAck* para enviar el ACK.

5.3.5. MergeBufferAdaptador

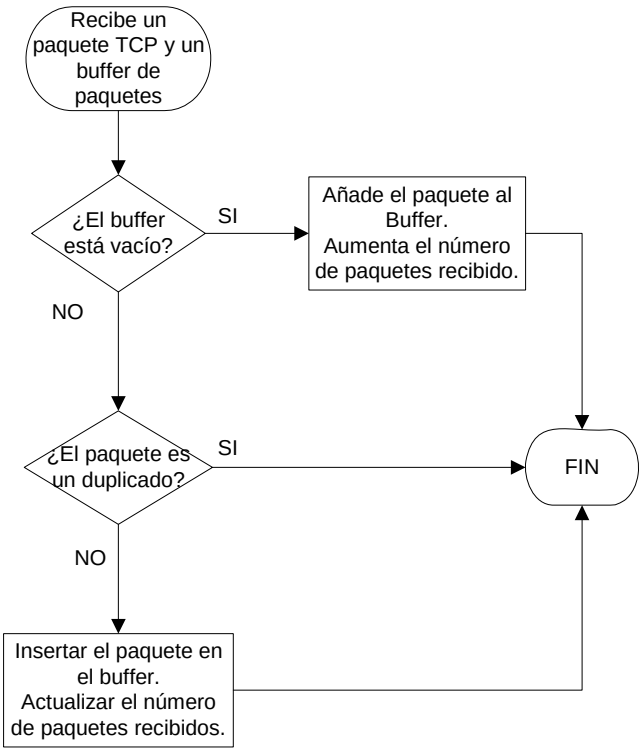


Figura 34 Diagrama de Flujo. MergeBufferAdaptador

- **Sinopsis**: Inserta un paquete en el buffer de la conexión.
- **Parámetros:** *HeaderPacket* * (entrada): paquete a insertar
  infoHash * (entrada): Información de la conexión y el buffer de paquetes
  BufferPkt * (salida): puntero al buffer ordenado
- **Descripción**: Almacena el paquete en el buffer siguiendo el orden indicado en el campo *usPacActual* y decrementa el número de paquetes a esperar.

Esta función se ejecuta únicamente desde la función *CheckPktTCPAdaptador*, cuando recibe un paquete de Datos Sc-Sc, actualiza el buffer y el número de paquetes esperados.

Si el buffer está vacío se comprueba que no se trate de un paquete ya recibido anteriormente y enviado al Capturador. Si es así puede que se haya perdido el ACK de confirmación AdO – AdO y por tanto se retransmite el ACK mediante la función *CreateSendAck_SC_SC*. Si no es un paquete perdido, se almacena en el buffer y finaliza la función.

Si el buffer ya contenía paquetes, se recorre hasta encontrar la posición relativa de acuerdo con el valor del campo *usPacActual*. Si algún paquete del buffer posee el mismo valor de ese campo, se trata de un duplicado y lo descarta. Los paquetes se ordenan de forma creciente, cuando encuentra su posición, se inserta y finaliza la función. A diferencia de los paquetes recibidos del Capturador, en este caso se permite la recepción no ordenada de los datos.

5.3.6. getNumSecuence

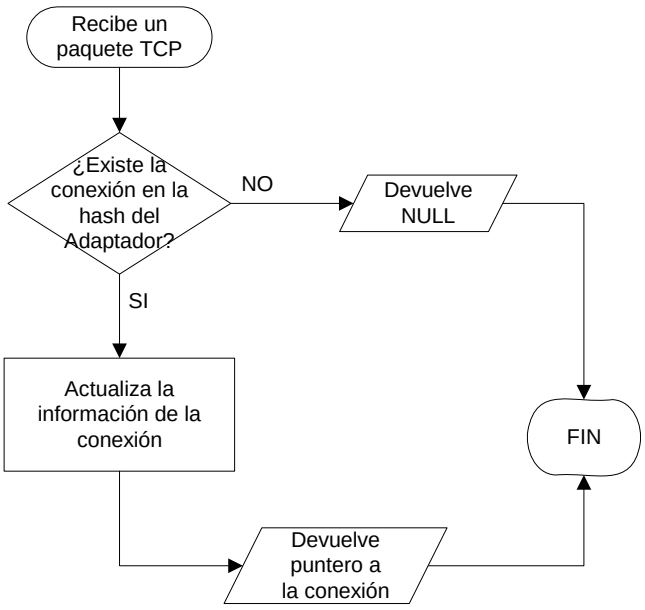


Figura 35 Diagrama de Flujo. getNumSecuence

- **Sinopsis**: Obtiene el Número de Secuencia del otro extremo de la conexión
- **Parámetros**: *HeaderPacket* * (entrada): paquete a analizar
  infoHash * (salida): Información de la conexión
- **Descripción**: Recorre la lista del Adaptador para encontrar la información correspondiente con el paquete.

Se llama a esta función desde *CheckPktTCPCapturador* cuando recibe un SYN-ACK, comprueba que sea la respuesta a SYN enviado previamente, y actualiza ambas listas con la información de número de secuencia de ambos extremos.

5.3.7. sendBuffer2Capturador

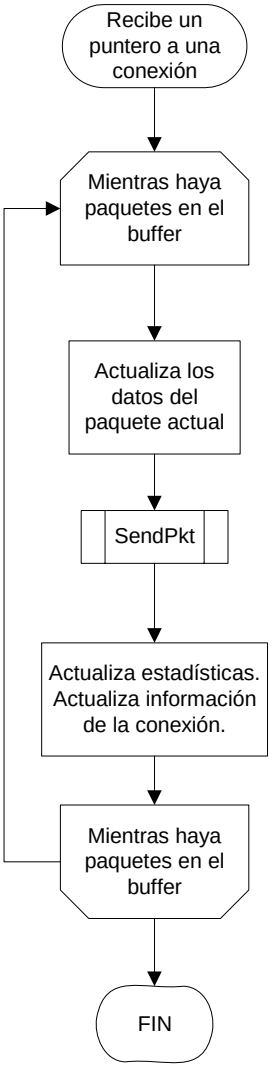


Figura 36 Diagrama de Flujo. sendBuffer2Capturador

- **Sinopsis**: Envía los paquetes almacenados al Capturador.
- **Parámetros**: *infoHash* * (entrada): Información de la conexión y el buffer a enviar.
  int (salida): Devuelve 0 si hemos enviado un FIN, 1 si no.
- **Descripción**: Envía los paquetes almacenados en el buffer al Capturador a ráfagas, modificando la información de los números de secuencia y asentimiento.

Esta función se invoca desde *CheckPktTCPCapturador* cuando se recibe un SYN-ACK. Existe una función similar llamada *sendBuffer2CapturadorDual* que se llama desde *CheckPktTCPAdaptador* cuando se han recibido todos los paquetes. La única diferencia es la lista de la que obtienen la información de los números de secuencia y asentimiento.

Si el primer paquete almacenado es un SYN lo ignora, ya que el SYN se envía mediante otra función. Recorre el buffer de paquetes y envía cada paquete al Capturador modificando el número de secuencia y asentimiento de acuerdo con la información almacenada en las listas, ya que al suplantar al otro extremo se eligen números al azar que no corresponden con los que establece el otro extremo.

Para intentar evitar saturar a la aplicación receptora, los paquetes se envían a ráfagas, en la primera ráfaga se envía el máximo de paquetes permitidos por la ventana de congestión de recepción, y en el resto de las ráfagas con la mitad de dicha ventana, esperando entre ráfaga y ráfaga 10 milisegundos por paquete enviado, de esta forma se intenta asegurar que la aplicación es capaz de procesar todos los datos.

Los paquetes se envían mediante una llamada a la función *SendPkt*.

5.3.8. sendBuffer2Redirector

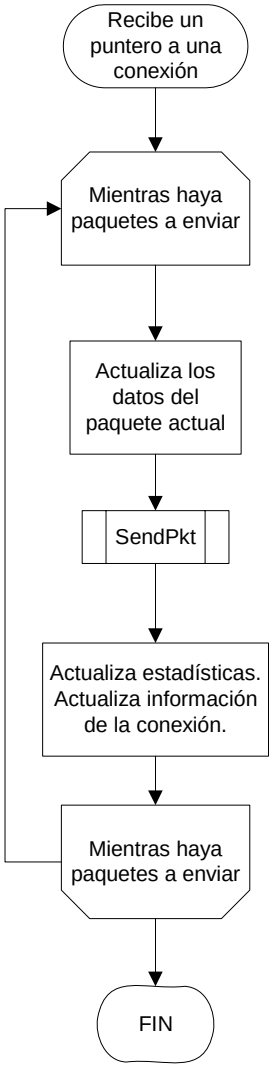


Figura 37 Diagrama de Flujo. sendBuffer2Redirector

- **Sinopsis**: Envía los paquetes almacenados al Redirector.
- **Parámetros**: infoHash *(entrada): Datos de la conexión y el buffer a enviar
  int (salida): 0 si ha habido un error, 1 si no hay errores
- **Descripción**: Recorre el buffer de paquetes, modifica los campos de la comunicación Sc-Sc y lo envía al Redirector.

Cuando hay que enviar paquetes de datos al AdO remoto se invoca esta función desde *CheckPktTCPCapturador* y desde *intData*.

Recorre el buffer de paquetes almacenados, y modifica en cada paquete los campos de la estructura *headerPacket* relativos a la comunicación Sc-Sc: *usNumPacTotal* y *usNumPacTotal.* Cada paquete se envía mediante la función *SendPkt*.

5.3.9. SendPkt

- **Sinopsis**: Envía un paquete al módulo destino indicado
- **Parámetros**: *AnsiString* (entrada): Destino al que enviar
  HeaderPacket * (entrada): Paquete a enviar
- **Descripción**: En función del destino (Capturador o Redirector), sincroniza los semáforos correspondientes y pide permiso para escribir en la memoria compartida, una vez lo ha hecho libera el semáforo.

Cada vez que se quiere intercambiar información con los otros módulos se llama a esta función, recibe un elemento de la estructura de intercambio con la información a transmitir y la información del módulo al que se quiere enviar el paquete, Capturador si es para la red local, o Redirector si es para el AdO remoto.

El intercambio de información se hace a través de memorias compartidas, por lo que es necesario el uso de semáforos para sincronizar la lectura/escritura en la memoria.

5.3.10. NewConectionCapturador /NewConectionAdaptador

- **Sinopsis**: Crea una nueva conexión en la lista del Capturador/Adaptador.
- **Parámetros**: *HeaderPacket* * (entrada): paquete con los datos de la conexión
  infoHash * (entrada): elemento dual de la otra lista.
- **Descripción**: Crea una nueva conexión en la lista del Capturador/Adaptador con la información del paquete recibido y se enlaza con su conexión dual de la lista del Adaptador.

Se llama a la función *NewConectionCapturador* desde *CheckPktTCPCapturador* para añadir un nuevo elemento en la lista de conexiones del Capturador.

Se recorre la estructura buscando si existe el origen, en cuyo caso sólo se añade la estructura que contiene la información del destino, y si no existe se crean ambas (origen y destino). Se enlaza con la entrada dual de la tabla del Adaptador (que hace referencia a esta misma conexión), de esta forma la información de la conexión es rápidamente accesible desde cualquier lista.

El comportamiento de la función *NewConectionAdaptador* es similar a esta, pero para la lista del Adaptador.

5.3.11. Funciones de búsqueda y retransmisión

En muchos casos es necesario realizar búsquedas, bien de una conexión, bien de un paquete. En este apartado incluimos todas estas funciones:

- *searchDirectConnection:* Cuando las funciones *CheckPktTCPCapturador* y *CheckPktTCPAdaptador* tienen que comprobar si existe una conexión en una de las listas que corresponda con un paquete dado, llaman a esta función, si el paquete tiene origen A y destino B, busca si existe una conexión con origen A y destino B. Para ello recorre la estructura de datos buscando el origen y si lo encuentra el destino.

- *searchReverseConnection:* Es la función dual a la anterior, en este caso busca una coincidencia en la lista, pero invirtiendo la información del origen y destino, es decir, si el paquete tiene origen A y destino B, busca si existe una conexión con origen B y destino A.

- *search_RTx:* Cuando se ejecuta la interrupción *int_SC_SC_Adaptador*, invoca a esta función para pedir retransmisión al otro extremo de los paquetes que aún no se han recibido. Para ello recorre el buffer de paquetes recibidos y con la información sobre el número total de paquetes en la ráfaga, comprueba qué paquetes no se han recibido y envía un NACK AdO- AdO al otro extremo por cada paquete para notificarle que inicie la retransmisión. Esto lo realiza mediante una llamada a la función *CreateSendAck_SC_SC*.

- *RtxPkt:* Cuando hay que retransmitir al otro extremo un paquete en concreto se llama a esta función, recorre el buffer de paquetes dados hasta encontrar el paquete solicitado, y lo envía al AdO remoto mediante la función *SendPkt*.

5.3.12. Funciones de Borrado

Se han implementado diferentes funciones de eliminación o borrado: de paquetes, de buffer, de conexión, etc., que resumimos en este punto.

- *DeleteConectionCapturador/DeleteConectionAdaptado:* Cuando se libera una conexión hay que eliminar la entrada correspondiente de la lista del Capturador y del Adaptador, por lo que recorre la lista buscando la entrada correspondiente, si existe, elimina el buffer de paquetes (mediante *DeleteBuffer*), mata los temporizadores y borra de la lista dicha conexión.

- *DeletePktBuffer*: Cuando se recibe un ACK de la aplicación local, se comprueba si hay paquetes esperando confirmación, y se comprueba si está asintiendo paquetes o solicitando retransmisión, en el primer caso se eliminarán del buffer los paquetes asentidos, y el segundo enviará de nuevo dichos paquetes.

Si el paquete confirmado por la aplicación local es un FIN, actualiza los estados de cierre de conexión y envía la confirmación al AdO remoto. Cuando se recibe la confirmación de todos los paquetes almacenados, se le notifica también al AdO remoto, enviando un ACK AdO-AdO.

- *DeleteBuffer*: Se invoca a esta función en dos situaciones: cuando se ha recibido un ACK Sc-Sc confirmando la recepción del buffer, es decir, la aplicación remota a recibido correctamente todos los paquetes, y cuando por algún motivo se liberan las conexiones y se borra toda la información de la conexión, incluido el buffer de paquetes.

En el primer caso se comprueba si el último paquete es un FIN, ya que significa que la aplicación remota ha confirmado el cierre de conexión en un extremo, se envía a la aplicación local la confirmación de cierre, se actualiza el estado de cierre de conexión, y se comprueba si está cerrada en los dos extremos, en cuyo caso se lanza el temporizador de cierre de conexión

- *DeleteHashCapturador/ DeleteHashAdaptador:* Cuando se produce el cierre del programa, en el caso de que hubiese alguna conexión establecida hay que liberar la conexión y la memoria, por lo que se recorre la lista y se elimina conexión a conexión, a cada conexión activa le envía un RST para abortar la conexión.

5.3.13. Funciones de creación de paquetes

Para la comunicación con la aplicación local es necesario que el AdO genere una serie de paquetes, como SYN, SYN-ACK, ACK o RST. Para la comunicación AdO-AdO también es necesario generar algunos paquetes, como ACK SC-SC o ACK reset.

Para ello se han creado una serie de funciones cuyo objetivo es generar y enviar estos paquetes. El funcionamiento es similar: reciben una estructura de intercambio, es decir, un paquete con la información TCP de la conexión, y la información adicional necesaria en cada caso, rellenan una estructura de intercambio con los parámetros adecuados y la envían al destino correspondiente mediante *SendPkt*.

Estas funciones son:

- *CreateSendAck_SC_SC*: Crea un ACK o un NACK Para la comunicación AdO-AdO. Necesita rellenar los campos específicos *usNumPacTotal* a 0 y *usPacActual* al valor pertinente.
- *CreateSendAckReset*: Confirma al AdO remoto la recepción de un RST en la comunicación AdO-AdO. Rellena los campos *usNumPacTotal* a 1 y *usPacActual* a 0.
- *CreateSendAckWin0*: Se usa para la limitación de capacidad, envía a la aplicación local un ACK con un tamaño de ventana win=0 o win= 4*MSS para activar o desactivar la recepción de paquetes.

- *CreateSendSynAck*: En el establecimiento de conexión, envía un SYN o un SYN-ACK a la aplicación local, con los flag activos según corresponda. El número de secuencia es elegido aleatoriamente.
- *CreateSendAck*: Envía a la aplicación local un ACK asintiendo los datos recibidos.
- *CreateSendReset*: Genera un paquete de RST para la aplicación o para el AdO remoto según los parámetros recibidos.

5.3.14. Interrupciones

Para controlar el tiempo de espera de un paquete, se lanzan diferentes temporizadores, asociados a unas rutinas o interrupciones.

- *IntData*: Cuando se reciben datos desde la aplicación local se lanza un temporizador de espera del siguiente paquete, si vence esta rutina actualiza envía el buffer almacenado al AdO remoto (*SendBuffer2Redirector*) y lanza un temporizador para esperar la respuesta del AdO remoto.
- *IntSYN*: Cuando el AdO, a petición del otro extremo, tiene que iniciar una conexión, envía un SYN y lanza un temporizador, si no se recibe la respuesta en el tiempo fijado, se vuelve a enviar el SYN a la aplicación, lanzándose de nuevo el temporizador. Si se supera el número de reintentos fijado, se envía un RST al AdO remoto y se lanza el temporizador para esperar la confirmación del AdO remoto.
- *IntSYNACK*: Cuando el AdO recibe un SYN de la aplicación local, le envía la respuesta SYN-ACK y lanza un temporizador para completar el establecimiento, si vence el temporizador sin recibir el ACK se elimina la conexión de la lista del Capturador.
- *IntFIN*: Según la RFC 793, tras cerrarse la conexión en ambos extremos, hay que esperar un tiempo antes de liberarla totalmente, y es lo que hace esta rutina, cuando el AdO detecta que se ha confirmado el cierre de conexión el los dos extremos lanza un temporizador, durante el cual los paquetes que lleguen serán ignorados. Cuando vence el temporizador esta rutina elimina la conexión de ambas listas.
- *Int_SC_SC_Capturador*: En la comunicación AdO-AdO, cuando se envía el buffer de paquetes al AdO remoto se lanza un temporizador para esperar la confirmación del otro extremo, si no recibe respuesta envía el último paquete almacenado en el buffer y vuelve a lanzar el temporizador. Si se supera el número de reintentos fijado se aborta la conexión, envía un RST a la aplicación local y elimina la conexión de ambas listas.
- *Int_SC_SC_Adaptador*: Cuando el AdO recibe datos del AdO remoto, lanza un temporizador para esperar el siguiente paquete, si no lo recibe esta rutina se encarga de solicitar la retransmisión de los paquetes que falten y se lanza de nuevo el temporizador. Tras superar el número de reintentos si la conexión estaba establecida con la aplicación se le envía un RST y se elimina de ambas listas.

- *IntRST*: Cuando se envía un RST al AdO remoto se espera a que confirme el paquete, por lo que se lanza un temporizador. Si vence se reenvía el RST y se lanza el temporizador, tras un número de reintentos, se elimina la conexión.
- *IntAbort*: Cuando durante un periodo lo suficientemente elevado no se produce ningún intercambio de paquetes en ninguno de los dos extremos se aborta la conexión eliminándola de las listas.
- *IntACK*: El AdO, tras enviar datos a la aplicación local, espera la recepción de ACK para confirmar dichos paquetes. Si no se recibe confirmación se reenvían los paquetes no confirmados, y tras un número de reintentos se aborta la conexión, enviando un RST al AdO remoto.

5.4. Análisis de resultados

Este Algoritmo se emplea para la transferencia de archivos desde un Servidor de Comunicaciones a otro cuando el medio de transmisión es, entre otros, radio, por lo que este es el escenario principal en las pruebas realizadas para conseguir los objetivos de este punto, que es realizar un análisis detallado de los resultados: transmitir paquetes TCP por un medio de banda estrecha. Se usaron terminales HF/VHF.

Figura 38 Escenario de Pruebas

Para la realización de la pruebas se diseñó una aplicación para enviar archivos entre dos equipos mediante el protocolo TCP. Dicha aplicación puede emplearse como emisor o receptor, según convenga. Como emisor solicita al usuario la dirección ip del destino y el archivo a enviar, divide el archivo en partes iguales al tamaño máximo del segmento TCP por defecto, y envía los datos a través de un socket. Como receptor la aplicación abre un socket y comienza a escuchar cuando se establece la conexión por cada paquete recibido envía la confirmación pertinente (ACK).

Se estudiaron diferentes tipos de envío, con diferentes tipos y tamaños de archivo (desde menores de 1Kb hasta mayores de 100Mb). Los casos analizados son:

- Envío normal Simple: De A a B.
- Envío normal Dual: De A a B y de B a A.
- Envío múltiple (Simple y Dual): Varios archivos de un extremo a otro.
- Envío varios terminales: Envío a varios destinos y recepción de varios orígenes.
- Sin Aplicación remota.
- Sin SC remoto.
- Fallo de Aplicación.
- Fallo del SC.

<u>* Contextos normales: Aplicación A y B funcionado correctamente.</u>

- Envío normal simple:

Desde el terminal A se envía un fichero hacia el terminal B. El caso más simple y más común, la transferencia de archivos de un terminal a otro. Se pudo comprobar que en todos los pasos de la conexión: establecimiento, envío de datos y cierre de conexión, el intercambio de segmentos era el esperado y el archivo se recibía correctamente.

- Envío normal dual:

El terminal A envía un archivo hacia el terminal B y al mismo tiempo el terminal B hacia el terminal A. Dos comunicaciones establecidas. Al igual que en el caso anterior, el intercambio de paquetes en todos los casos era correcto y los archivos se recibían correctamente.

- Envío múltiple simple y dual:

Desde el terminal A se envían varios archivos simultáneamente hacia el terminal B, mientras que el terminal B puede o no enviar uno o varios archivos hacia el terminal A. Se comprobó que el comportamiento era el esperado para comunicaciones TCP y que los archivos se recibían correctamente.

- Envío varios terminales:

Se repitieron los casos anteriores con más de un terminal receptor y con más de un terminal emisor, funcionando todo correctamente.

Para asegurar que el comportamiento en el caso de pérdidas de paquetes era correcto, se introdujo una modificación en el código que descartaba paquetes aleatoriamente entre Aplicación-AdO, en la comunicación AdO-AdO, y entre AdO-Aplicación. De esta forma se pudo observar que en caso de pérdida de diferentes tipos de paquetes, las peticiones de retransmisión, y las retransmisiones funcionaban correctamente.

Todos estos casos se repitieron con distintos tamaños de archivos y distintos límites de capacidad. En los casos que se superaba el límite de congestión, el AdO cerraba la ventana y la aplicación dejaba de enviar datos, cuando se recibía la confirmación del AdO remoto y se liberaba el buffer, le abría la ventana y la aplicación continuaba con la transferencia del archivo, recibiéndose correctamente en el otro extremo.

* Contextos especiales

Después de probar diferentes escenarios con situaciones normales, se forzaron situaciones anómalas para comprobar el comportamiento del Algoritmo.

En el primer caso de análisis la aplicación A funciona correctamente y la aplicación B no se encuentra lanzada. El AdO A suplanta al extremo B, se establece la conexión, y la Aplicación A envía los paquetes de datos e inicia el cierre de conexión. El AdO A envía los paquetes al AdO B, y este último intenta establecer la conexión con la aplicación B, que al no estar ejecutándose no responde al SYN. Tras vencer los temporizadores y superarse el número de reintentos, el AdO B envía un RST al AdO A, y este a la aplicación, abortando la conexión. Se comprueba que todo el intercambio de paquetes en ambos extremos de la conexión es correcto y el comportamiento es el esperado.

Un caso similar al anterior, en este caso el AdO remoto no está lanzado. La comunicación entre la aplicación y el AdO es la correcta, y el AdO A envía el buffer al AdO B, al no recibir respuesta procede a enviar el último paquete de nuevo, tras superarse el número de reintentos, envía un RST a la aplicación y se aborta la conexión. El intercambio de paquetes en todo el proceso es correcto y el comportamiento es el esperado.

Otro de los casos consistió en matar los procesos de la aplicación A y B en diferentes puntos de la conexión:

- En el establecimiento de la conexión entre la aplicación A y el AdO A. Al no recibir el ACK para completar el establecimiento se borra la conexión.
- Durante el envío de datos de la aplicación A al AdO A. El AdO A interpreta que la aplicación no va a enviar más paquetes, por lo que envía el buffer al otro extremo, recibiendo la confirmación, y al no haber intercambio de paquetes durante un periodo de tiempo elevado se aborta la conexión, enviando un RST al otro extremo.
- En el establecimiento de la conexión entre el AdO B y la aplicación B. Tras varios intentos de establecimiento, aborta la conexión enviando un RST al otro extremo.
- En el envío de datos del AdO B a la aplicación B. Al no recibir confirmación, el AdO B reenvía los datos, y tras cierto número de reintentos aborta la conexión enviando un RST al otro extremo.
- En el cierre de conexión, al igual que en el caso de datos, al no recibir respuesta se aborta la conexión.

Se comprobó que al matar la aplicación y volverla a lanzar mientras el AdO permanecía con los reintentos, la aplicación no reconocía los paquetes pertenecientes a ninguna conexión suya y envía un Reset, que era tratado adecuadamente por el AdO.

De igual forma, se procedió al cierre forzoso de un AdO en diferentes puntos de la conexión, y el intercambio de paquetes, temporizadores, interrupciones y tratamiento de reset fue el esperado.

También se realizaron pruebas con la aplicación “iperf”, que es una herramienta cliente/servidor que permite generar flujo de datos TCP o UDP para medir el rendimiento de la red. Permite configurar varios parámetros, como el tamaño de la ventana en TCP o la longitud de los paquetes. También permite el envío de ficheros. Se realizaron diferentes pruebas, variando los parámetros de configuración, y enviando ficheros, y se comprobó que se recibían correctamente los paquetes, y en el caso de envío de archivos que llegaban correctamente al otro extremo.

En último lugar se realizaron pruebas con una aplicación FTP. Se establece la conexión y se consigue enviar archivos pequeños (menos de 1Mb), pero con tamaños mayores se observa que aunque a nivel TCP el estado de la comunicación es correcto, a nivel de la aplicación FTP vencen los temporizadores y reinicia la conexión, por lo que no se completa el envío.

CAPÍTULO 6:
Memoria Económica.

6. Memoria Económica

En este capítulo se realizará una estimación del coste de la ejecución del presente proyecto. Se desglosará el coste total en varios grupos que cubran todos los gastos derivados de la ejecución.

6.1. Coste de materiales.

En primer lugar se analizarán todos los costes de material, incluyendo los gastos necesarios para el soporte informático utilizado, tanto software como hardware, y el conjunto de materiales de oficina empleados.

- **Gastos de productos hardware.**

Para el desarrollo y la fase de depuración se emplearon tres equipos, en cuyo coste se van a incluir los periféricos necesarios como monitor, teclado y ratón.

Los equipos y terminales utilizados en la fase final del proyecto fueron proporcionados por el cliente y no se van a incluir como gasto directo.

Ordenador Pentium IV, 1.8GHz	800 €
Ordenador Pentium IV, 2.66GHz	950 €
Ordenador portátil Acer, 1.73GHz	380 €
Total	**2130 €**

- **Gastos de productos software.**

Se incluyen las licencias de las herramientas utilizadas para el desarrollo del proyecto.

Licencia Microsoft Windows XP SP2, 3 unidades	297 €
Microsoft Office 2003	127 €
Visio 2000	60 €
C++ Builder 6 Professional Educational Edition	118 €
Total	**602 €**

- **Otros gastos.**

Además de los gastos de material de oficina, agrupados en recursos consumibles y no consumibles, se incluyen en este apartado los costes indirectos como pueden ser la conexión ADSL, la luz, etc., calculados como un 2% de los costes directos.

Material de oficina consumible	160 €
Material de oficina no consumible	100 €
Costes indirectos (ADSL, etc.)	555 €
Total	**815 €**

Teniendo en cuenta estos 3 aspectos, el coste total de materiales asciende a **tres mil quinientos cuarenta y siete euros: 3547 €**.

6.2. Coste de personal.

Para realizar una estimación de los gastos de personal nos hemos basado en el salario medio de un diplomado contratado por la Universidad Carlos III de Madrid, que durante el año 2009 se encuentra aproximadamente en 28000€ al año, unos 2300 € al mes.

El tiempo dedicado al desarrollo del presente proyecto es de aproximadamente 8 meses, incluyendo las fases de investigación previa, estudio del sistema completo y su relación con el algoritmo, desarrollo, depuración, pruebas y documentación, siendo muy complicado discernir el tiempo empleado a cada tarea concreta.

Con estos datos, el coste de personal ascendería a **dieciocho mil cuatrocientos euros, 18400 €.**

6.3. Coste total de ejecución.

La suma de los costes anteriores nos indica el coste total de la ejecución, que en este caso asciende a **veintiún mil novecientos cuarenta y siete, 21947 €**.

A este importe se le debe aplicar el Impuesto sobre el Valor Añadido (IVA) vigente de un 16%, resultando en el siguiente coste final:

Coste de ejecución sin IVA	21947 €
IVA (16%)	3511,52 €
Coste final con IVA	**25458,52 €**

El coste final asciende a **veinticinco mil cuatrocientos cincuenta y ocho euros con cuarenta céntimos**.

CAPÍTULO 7:
Conclusiones y líneas futuras.

7. Conclusiones y líneas futuras

7.1. Conclusiones.

El Servidor de Comunicaciones es un gestor de encaminamiento y uso alternativo de los medios de transmisión existentes. El cliente proporcionó tanto los drivers como los terminales para HF/VHF, que poseían unas características concretas por lo que el Servidor de Comunicaciones tenía que adaptarse a ellas. En las especificaciones y requerimientos venía recogida la necesidad de establecer sesiones TCP a través de HF/VHF, orientadas a la transferencia de archivos entre un Servidor de Comunicaciones y otro.

Tras las pruebas realizadas, se comprueba que en el marco establecido el Algoritmo de Optimización funciona correctamente, que el comportamiento a nivel TCP cumple con las normas recogidas en la RFC 793 y que el intercambio de paquetes en los diferentes estados de una conexión TCP es el adecuado.

En Algoritmo de Optimización cumple los requisitos expuestos por el cliente: establecer conexiones TCP entre sus terminales HF/VHF y transferir archivos, en este sentido se puede considerar un éxito, ya que se comprueba que en el mismo escenario sin utilizar el AdO no se logra establecer la conexión.

Sin embargo también se pudo comprobar que presenta ciertas limitaciones debidas a los requisitos de algunas aplicaciones, ya que en caso de FTP, aunque a nivel TCP esté funcionando correctamente, no siempre se consigue realizar el intercambio de archivos debido a los temporizadores de la aplicación FTP.

7.2. Mejoras futuras.

Para finalizar, se presentan una serie de posibles mejoras si se continúa con el desarrollo del proyecto.

En lo que respecta al Algoritmo en sí, hay unos cuantos aspectos a desarrollar, como pueden ser:

- Negociación de las opciones TCP. Es posible negociar entre los dos extremos algunas opciones, como el tamaño máximo de segmento o las confirmaciones selectivas. En esta versión se descartó esta opción dada la complejidad, ya que si se aceptan las opciones en un extremo nada nos garantiza que el otro extremo vaya aceptarlas, lo que dificultaría el intercambio de paquetes, pero es una opción a estudiar en futuros desarrollos.
- Implementación Keep - Alive. No se incluyó en esta versión ya que no es una práctica globalmente aceptada y no todas las aplicaciones lo soportan. Según las indicaciones de la RFC el tiempo por defecto no debe ser menor de dos horas. En

futuras versiones se podría incluir, teniendo en cuenta que es la aplicación la que indica si está activo o no (en el caso de estar implementada).

- Controlar la ventana de congestión de forma dinámica en el envío de datos AdO-Aplicación. Este es seguramente el principal aspecto a mejorar de Algoritmo. Actualmente no se verifica el tamaño de ventana de la aplicación receptora, ya que si esta la cierra no envía el ACK correspondiente y transcurrido un tiempo se volverían a enviar los datos no confirmados, por lo que los datos no se perderían. Pero es un comportamiento importante para ser implementado en el próximo desarrollo.

Con respecto al Algoritmo dentro del Servidor de Comunicaciones, que el usuario pueda elegir activar o desactivar la optimización. No es algo trivial, ya que no le afecta sólo a él si no a los equipos remotos con los que se comunique, por lo que necesitaría sincronizarse para activarlo / desactivarlo.

También se puede estudiar que al igual que en TCP, entre dos AdO se negocien o pongan en común sus parámetros de configuración, como el límite del buffer o el tiempo de espera.

Para el caso de la aplicación FTP una posible solución sería utilizar aplicaciones propietarias, o aquellas que permitan un mayor configuración de los temporizadores en función de las características del medio de transmisión empleado.

Anexos

8. Anexos

8.1. Glosario de términos

ACK: *Acknowledgement*

AdO: *Algoritmo de Optimización.*

HF: *High Frequency*

IP: *Internet Protocol*

LAN: *Local Area Network*

MSS: *Maximum Segment Size*

RFC: *Request For Comments*

RST: *Reset*

SC: *Servidor de Comunicaciones*

SYN: *Synchronize*

TCP: *Transmission-Control-Protocol*

UDP: *User Datagram Protocol*

VHF: *Very High Frequency*

WAN: *Wide Area Network*

8.2. Estructura de intercambio

A continuación se muestra la definición de la estructura HeaderPacket:

```
typedef      struct HeaderPacket {
 // Cabecera ethernet.
unsigned char ucMacSource[ETH_ALEN];
unsigned char ucMacDestination[ETH_ALEN];
unsigned short usProto;

 // Cabecera paquete IP.
// Versión del protocolo
unsigned char ucVersion;
// Tamaño de la cabecera sin datos
unsigned char ucLongHeader;
// Tipo de servicio
unsigned char ucTypeServive;
// Precedencia
unsigned char ucPrecedence;
// TOS tipo de servicio.
unsigned char ucTOS;
// MBZ
unsigned char ucMBZ;
// Longitud total del datagrama
unsigned short shTotalLength;
// Identificador único
unsigned short usIdentification;
// Flags reservado
unsigned char ucFlagReserved;
// Flags de indicación de fragmentación.
unsigned char ucFlagFragment;
// Flag de indicación de mas fragmentos
unsigned char ucFlagMoreFragment;
// Offset
unsigned short scFragmentOffset;
// global Offset, los campos de flag mas el fram offset.
unsigned short scGlobalFragmentOffset;
// TTL
unsigned char ucTTL;
// Tipo de protocolo
unsigned char ucProtocol;
// Checksum cabecera
unsigned short usHeaderChecsum;
// Dirección origen
unsigned long ulIPSource;
// Direccion destino
unsigned long ulIPDestination;
// longitud de las opciones
unsigned char ucOptionsLenght;

// Cabecera paquete TCP.
// Puerto origen
unsigned short usPortSourceTCP;
// Puerto destino
unsigned short usPortDestinationTCP;
// número de secuencia
unsigned long ulSecuenceNumber;
// número de Ack.
unsigned long ulAckNumber;
// offset de los datos.
unsigned char ucDataOffset;
// flag urgencia.
unsigned char ucFurg;
// flag Ack
```

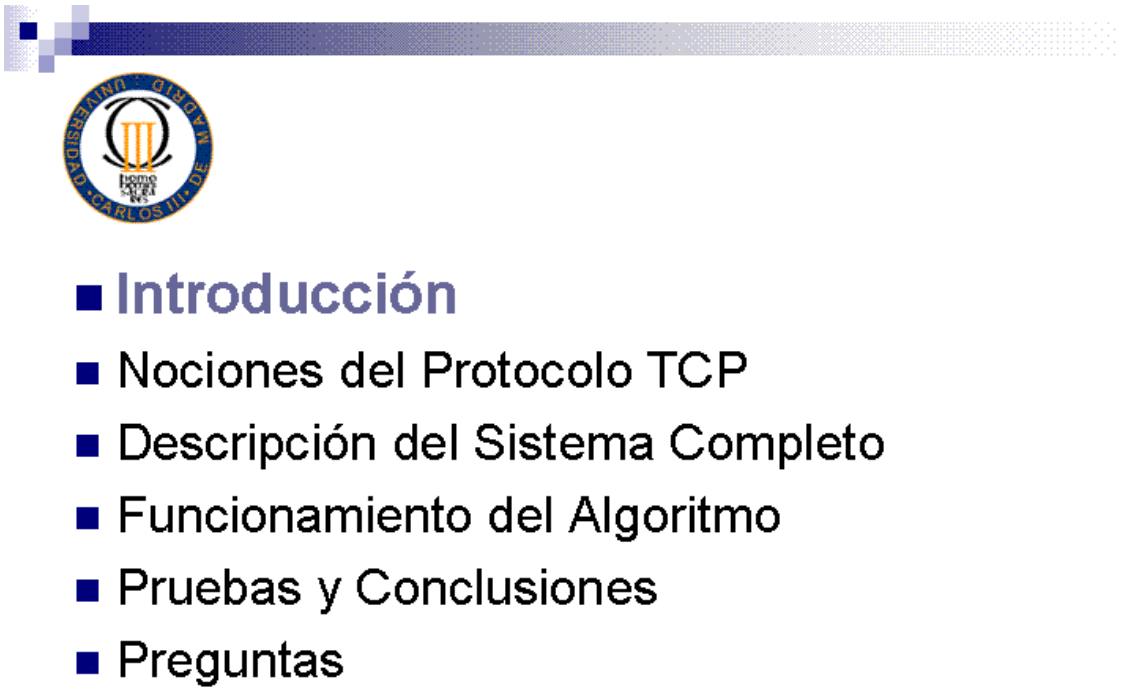
```
unsigned char ucFack;
// flag push
unsigned char ucFpush;
// flag reset conexión.
unsigned char ucFrstConnect;
// flag de sincronia.
unsigned char ucFsinc;
// flag fin datos.
unsigned char ucFdataEnd;
// campo con todos los flags
unsigned char ucGlobalFlagsTCP;
// ventana de desplazamiento W
unsigned short usWin;
// checksum
unsigned short usChecksunTCP;
// urgencia
unsigned short usUrgence;

// Cabecera paquete UDP.
// Puerto origen
unsigned short usPortSourceUDP;
// Puerto destino
unsigned short usPortDestinationUDP;
// tamaño
unsigned short usLengthUDP;
// checksum
unsigned short usChecksunUDP;

// Control de paquetes TCP
// número total de paquetes que forman la trama TCP
unsigned short usNumPacTotal;
// número de paquete de la trama
unsigned short usPacActual;

// Datos
unsigned short uiLengthData;
unsigned char cData[MAX_BUFF_DATA];
}HeaderPacket;
```

8.3. Presentación



Introducción

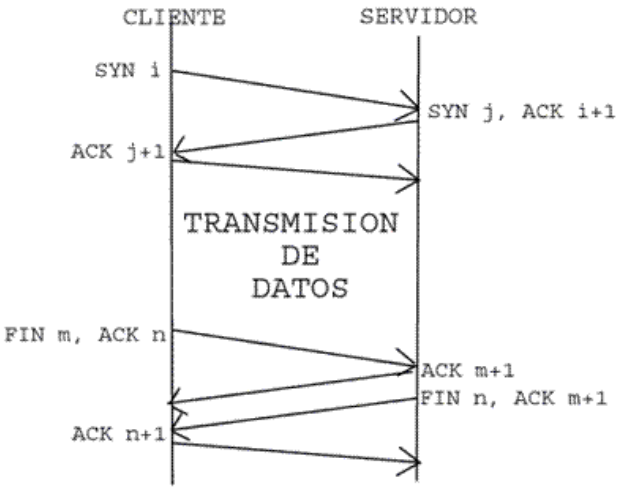
- Sistema Global de Comunicaciones
- TCP exige confirmación de datos
- Canales de Banda estrecha
- Retardos y tiempos de espera
- Comunicación necesaria

- Introducción
- **Nociones del Protocolo TCP**
- Descripción del Sistema Completo
- Funcionamiento del Algoritmo
- Pruebas y Conclusiones
- Preguntas

TCP

- Protocolo del nivel de transporte
- Orientado a conexión
- Control de Flujo
- Full-Duplex
- Recuperación frente a errores
- Asentimientos ACK's
- Retransmisión de paquetes

TCP: Ejemplo



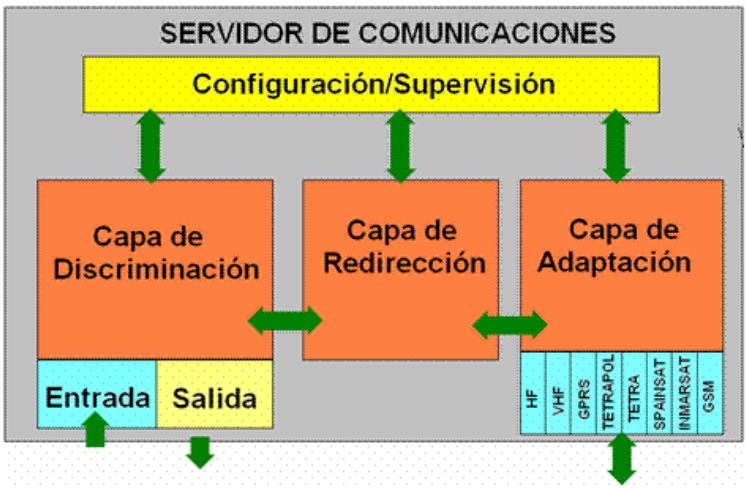
- Introducción
- Nociones del Protocolo TCP
- **Descripción del Sistema Completo**
- Funcionamiento del Algoritmo
- Pruebas y Conclusiones
- Preguntas

Servidor de Comunicaciones I

- Gestor de encaminamiento
- Diferentes redes de transmisión:

  IP-ETHERNET, INMARSAT, GSM/GPRS, TETRAPOL, TETRA, THURAYA, VHF y HF
- Modular
- Transmisión TCP

Servidor de Comunicaciones II

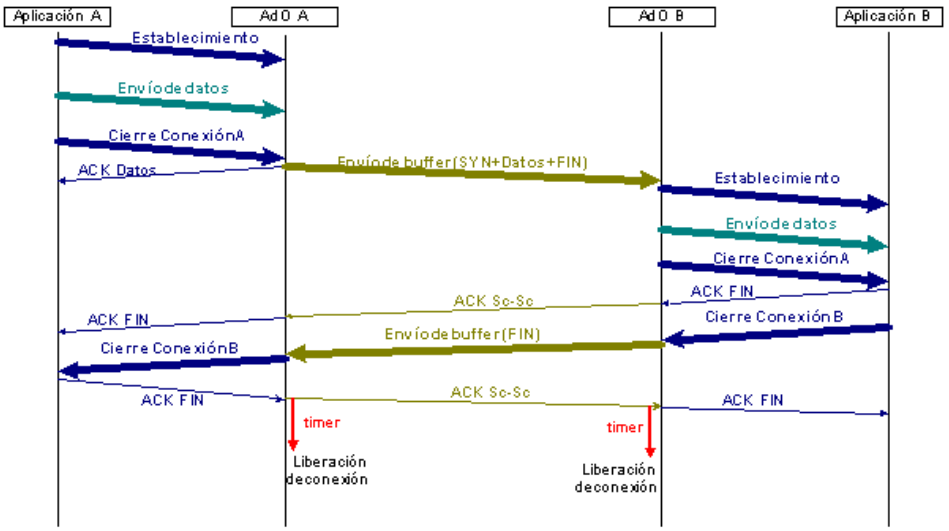


- Introducción
- Nociones del Protocolo TCP
- Descripción del Sistema Completo
- **Funcionamiento del Algoritmo**
- Pruebas y Conclusiones
- Preguntas

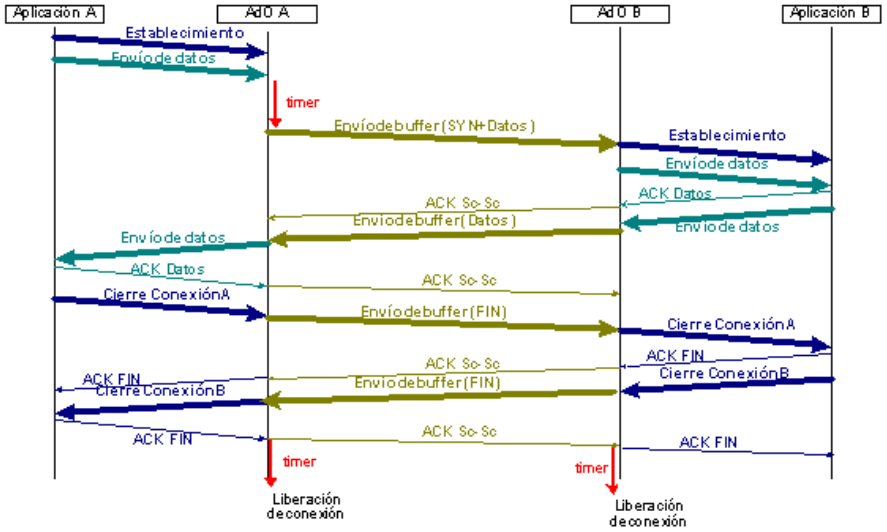
Algoritmo de Optimización

- Establecer comunicaciones TCP en redes de Banda Estrecha
- Suplantación identidad Origen y Destino
- Validación local de las tramas
- Comunicación Sc-Sc

Ejemplo de funcionamiento I



Ejemplo de funcionamiento II

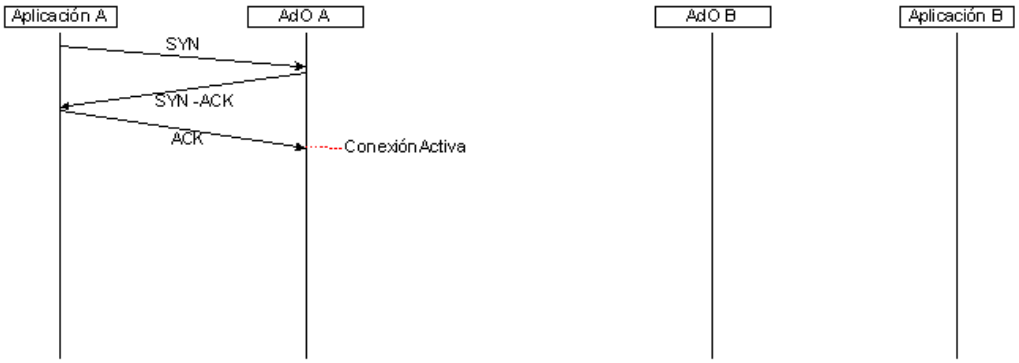


Pasos de una conexión

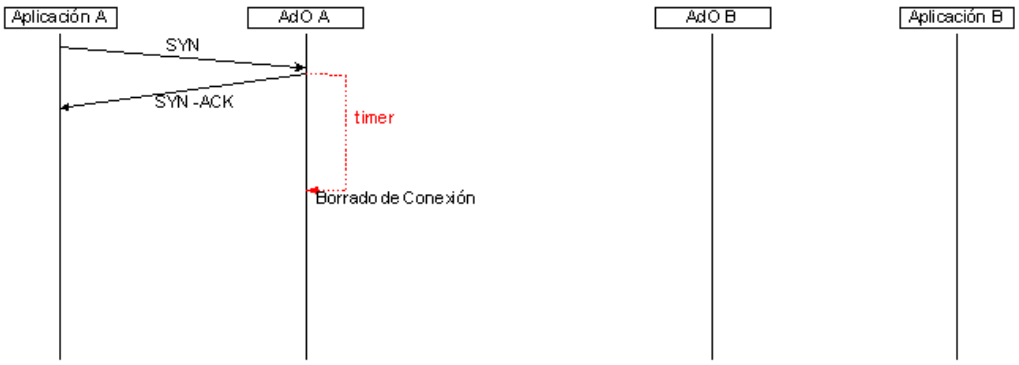
- Establecimiento de conexión Aplicación – AdO .
- Envío de datos Aplicación – AdO.
- Envío de datos AdO – AdO.
- Establecimiento de conexión AdO – Aplicación.
- Envío de datos AdO – Aplicación.
- Cierre de conexión.

- RST
- Limitación de Capacidad

Establecimiento de conexión I



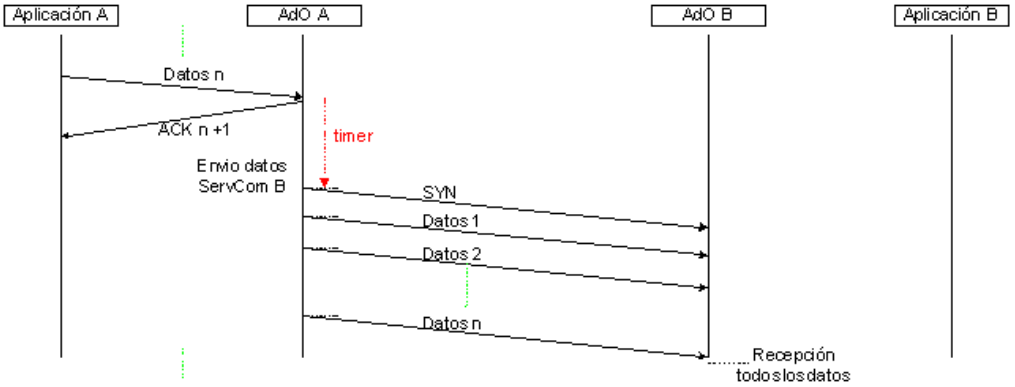
Establecimiento de conexión II



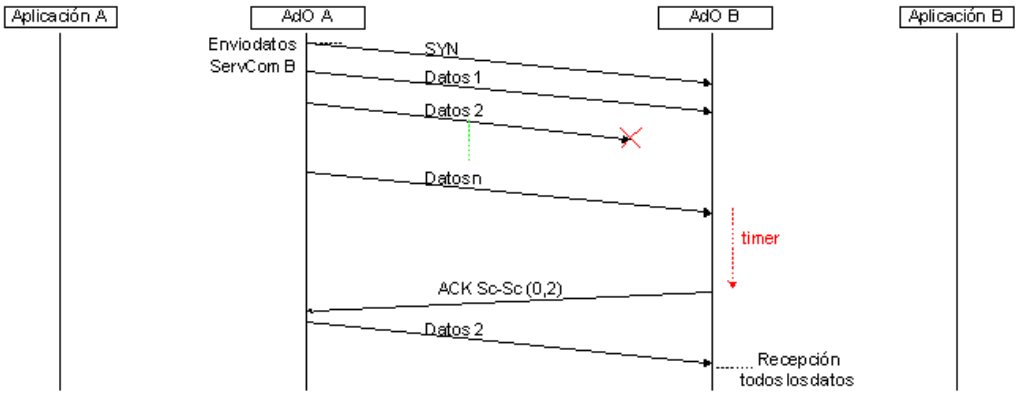
Datos Aplicación – AdO I

Datos Aplicación – AdO II

Datos AdO – AdO I

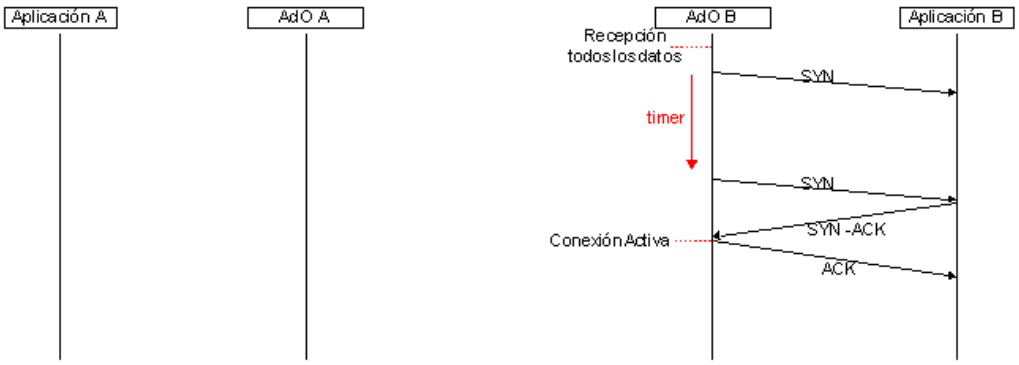


Datos AdO – AdO II

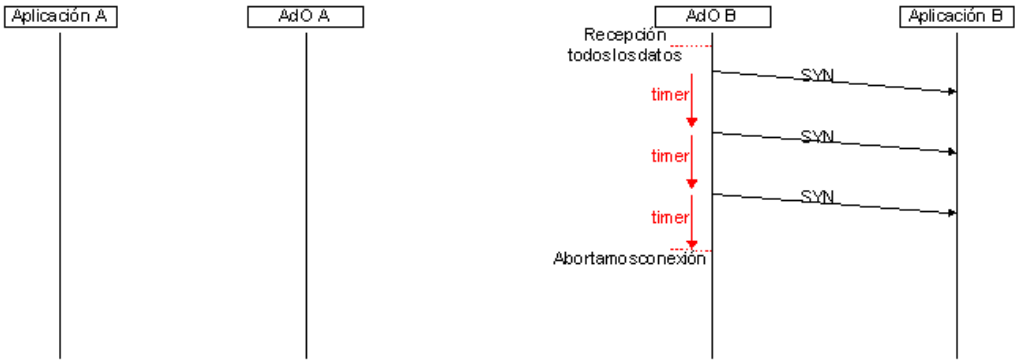


Inicio de conexión remota I

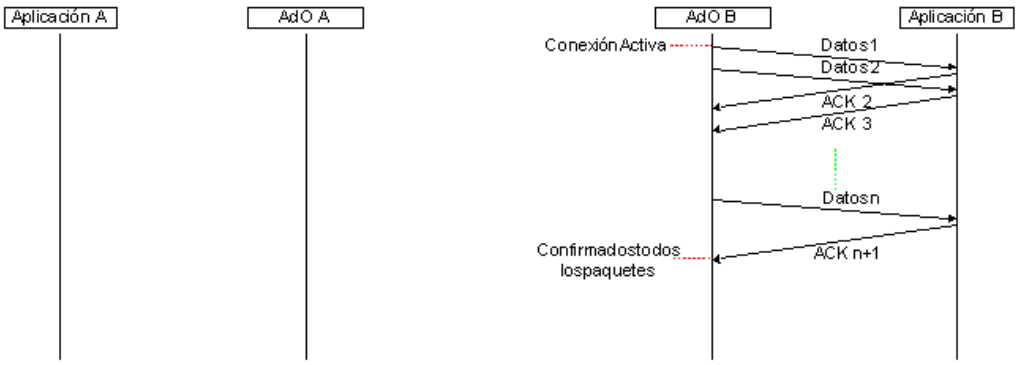
Inicio de conexión remota II



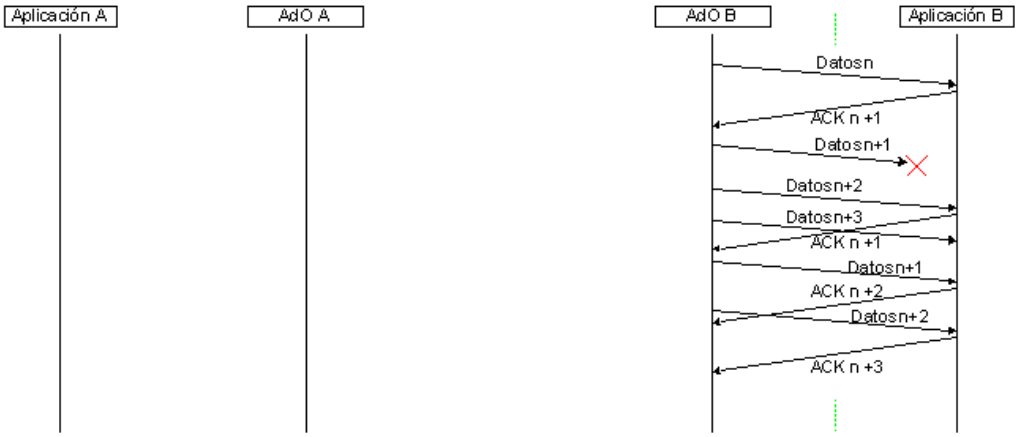
Inicio de conexión remota III



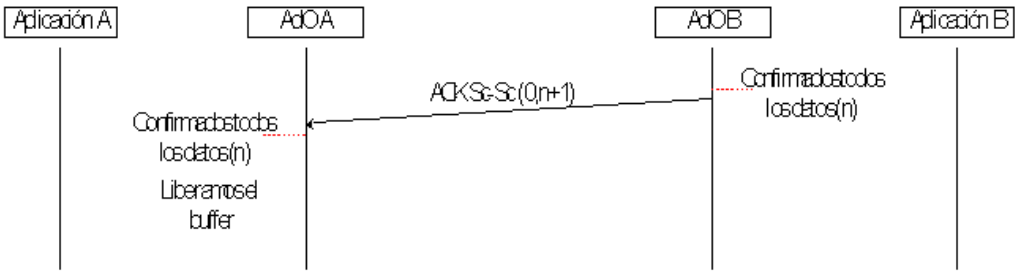
Datos AdO – Aplicación I



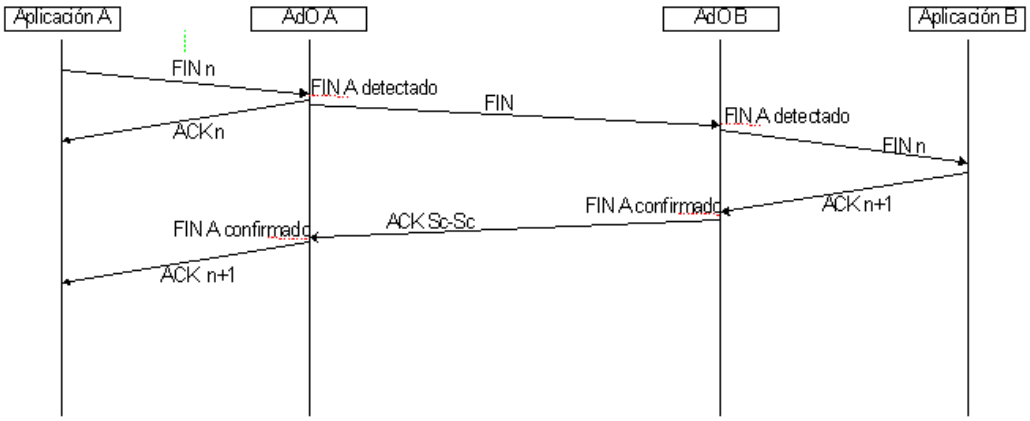
Datos AdO – Aplicación II



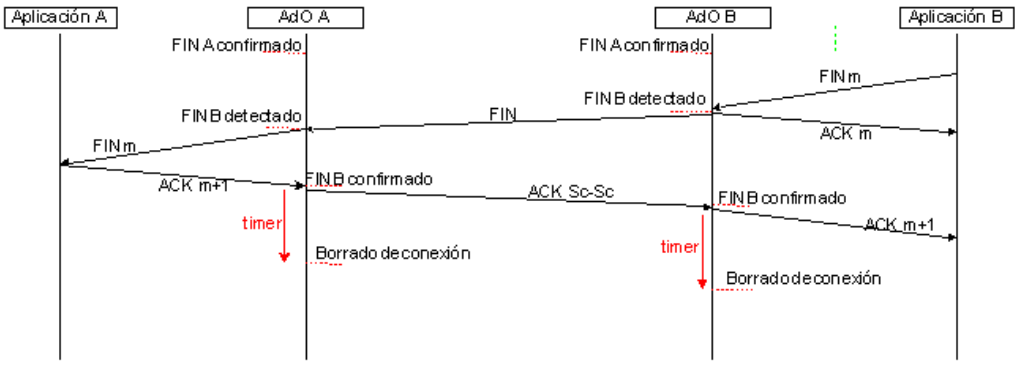
Confirmación AdO - AdO



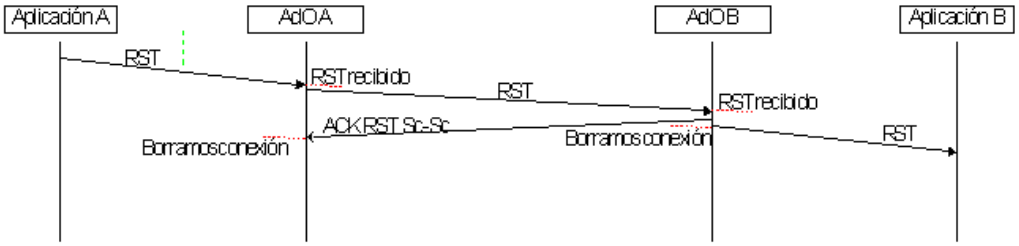
Cierre de conexión I



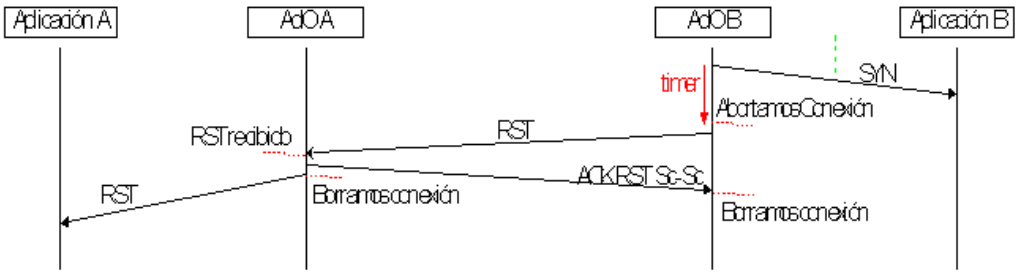
Cierre de conexión II



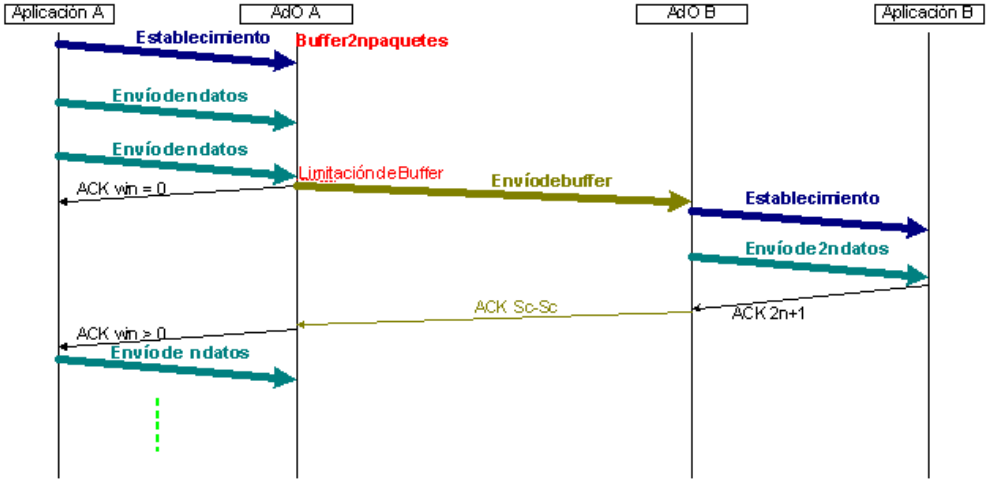
Tratamiento de RST I



Tratamiento de RST II



Limitación de Capacidad



- Introducción
- Nociones del Protocolo TCP
- Descripción del Sistema Completo
- Funcionamiento del Algoritmo
- **Pruebas y Conclusiones**
- Preguntas

Escenario de Pruebas

Pruebas realizadas

- Envío normal Simple: De A a B.
- Envío normal Dual: De A a B y de B a A.
- Envío múltiple: Varios archivos.
- Envío/Recepción varios terminales
- Sin Aplicación remota.
- Sin SC remoto.
- Fallo de Aplicación.
- Fallo del SC.

Conclusiones y Mejoras

- Envío de ficheros Sc - Sc
- Conexiones TCP a través de HF/VHF
- RFC 793

- Negociación de opciones, Keep – Alive, ventana de congestión

- Introducción
- Nociones del Protocolo TCP
- Descripción del Sistema Completo
- Funcionamiento del Algoritmo
- Pruebas y Conclusiones
- **Preguntas**

8.4. Código implementado

```
/*********************************************************************
*                    CODIGO DE GESTION TCP                          *
*********************************************************************/

/*****************************************************************************************************
NOMBRE:      CheckPktTCPCapturador
SINOPSIS:    Recibe un paquete del Capturador y lo analiza en funcion del
             protocolo TCP.
PARAMETROS:  HeaderPacket * (entrada): Pkt recibido
DESCRIPCION: Comprueba que tipo de paquete ha recibido: SYN, SYN-ACK, FIN, RST,
             Datos..., si dicho paquete es valido, si pertenece a una conexion
             ya establecida, y en funcion de esto le contesta siguiendo los
             pasos del protocolo TCP establecidos en la RFC 793
*****************************************************************************************************/

int CheckPktTCPCapturador(HeaderPacket *pktTCP){

   hashTable *auxHash, *newHash;
   infoHash  *endInfo, *newInfo, *auxInfo, *info2Capt;
   bufferPkt *auxBuffer, *bufferOrdenado;
   DWORD thId_Cap = 0, thId_Red = 0;
   int ackAdaptador;

   printf("(CheckPktTCPCapturador) paquete recibido\n");
   // Tratamiento de paquetes SYN ACK para establecer conexiones del adaptador
   if (pktTCP->ucFsinc == (unsigned char)1  &&  pktTCP->ucFack == (unsigned char)1){
      printf("(CheckPktTCPCapturador) SYN-ACK\n");
      if (pktTCP->usIdentification != Ident){
         if (gHashAdaptador != NULL) {
            //Comprobar que corresponda a una conexion del adaptador
            info2Capt = getNumSecuence(pktTCP);
            if (info2Capt != NULL) {
               timeKillEvent(info2Capt->timerID);
               if (info2Capt->conexDual == NULL) {
                  printf("Nueva conexion en el Capturador\n");
                  NewConectionCapturador(pktTCP,info2Capt);
                  MyStat->numConexTCPactivas++;
                  MyStat->numConexTCPtratadas++;

                  //Tras obtener el nº de secuencia, envio el ACK y los datos
                  CreateSendSynAck(pktTCP,info2Capt->numSecPkt,0,1,true,fileNameEnv);
                  if (info2Capt->buffer != NULL){
                     if (info2Capt->buffer->pkt.ucFsinc == (unsigned char) 1) {
                        if (info2Capt->buffer->next == NULL) {
                           CheckAckAdaptador(pktTCP);
                           return 0;
                        }
                     }
                     printf("(CheckPktTCPCapturador) lanzo hilo SendBuffer2Capturador\n");
                     info2Capt->hiloCapturador =
CreateThread(NULL,0,(LPTHREAD_START_ROUTINE)SendBuffer2Capturador,info2Capt,0,&thId_Cap);
                     //info2Capt->hiloCapturador = thId_Cap;
                  }
               }
            }
            else{
               //Recibimos un Syn-Ack sin haber enviado nosotros un SYN
               //Mandar un reset
               printf("Recibido SYN-ACK sin mandar un SYN.... Enviando RST\n");
               CreateSendReset (pktTCP, 1, fileNameEnv);
            }
         } else {
            //Recibimos un Syn-Ack sin haber enviado nosotros un SYN
            //Mandar un reset
            printf("Recibido SYN-ACK sin mandar un SYN.... Enviando RST\n");
            CreateSendReset (pktTCP, 1, fileNameEnv);
         }
      }
      return 0;
   }
```

```
   // -  Si recibimos un ACK para el adaptador (capturador->adaptador) eliminamos
   //    los paquetes asentidos del buffer, lo hace CheckAckAdaptador.
   // -  Si el ACK es para el redirector(capturador->redirector) lo almacenamos
   //    y vemos si existe una conexion ya
   if ((pktTCP->uiLengthData - (pktTCP->ucDataOffset * 4)) == 0){ //No hay datos
      if (pktTCP->ucFack == 1 && pktTCP->ucFdataEnd != 1 && pktTCP->ucFrstConnect != 1){
         printf("(CheckPktTCPCapturador) ACK\n");
         //Es un ACK pero no es un FIN ni un RST
         ackAdaptador = CheckAckAdaptador(pktTCP);
         if (ackAdaptador != 0) {
            return 0;
         }
         else {
            //Si no existe una conexion en el adaptador, debemos comprobar que exista
            //en el capturador
            info2Capt = searchDirectConnection(pktTCP, gHashCapturador);
            if (info2Capt != NULL) {
               timeKillEvent(info2Capt->timerID);
               printf("Nueva conexion en el Adaptador\n");
               NewConectionAdaptador(pktTCP, info2Capt);
               info2Capt->timerID = timeSetEvent(TIME_OVER, 0,IntData, (unsigned long)info2Capt, TIME_ONESHOT);
               MyStat->numConexTCPactivas++;
               MyStat->numConexTCPtratadas++;
            }else {
               //un ack para una conexion que no existe... enviar un rst
               printf("Recibido ACK sin mandar un SYN.... Enviando RST\n");
               CreateSendReset (pktTCP, 1, fileNameEnv);
            }
            return 0;
         }
      }
   }

   // La hash no existe => creamos la hash y añadimos el paquete.
   if (gHashCapturador == NULL){
      printf("Hash nula\n");
      // Verificamos si es una petición de establecimiento de comunicacion
      if ((pktTCP->ucFsinc != (unsigned char)1)){
         if (pktTCP->ucFrstConnect != 1) {
            printf("Recibido Pkt sin mandar un SYN.... Enviando RST\n");
            CreateSendReset (pktTCP, 1, fileNameEnv);
         } else {
            //Buscar en la hash del adaptador y si existe, borrar conexion
            printf("RST recibido\n");
            auxInfo = searchReverseConnection(pktTCP, gHashAdaptador);
            if (auxInfo != NULL) {
               //Estamos ante un rst como respuesta a un syn -> mandar reset al otro extremo
               timeKillEvent(auxInfo->timerID);
               printf("Recibido RST como respuesta a un SYN.... Enviando RST al otro SC(Redirector)\n");
               MyStat->pktSndRed++;
               MyStat->bytSndRed = MyStat->bytSndRed + pktTCP->uiLengthData - (pktTCP->ucDataOffset * 4);
               MyStat->pktSndRedTCP++;
               MyStat->bytSndRedTCP = MyStat->bytSndRedTCP + pktTCP->uiLengthData - (pktTCP->ucDataOffset * 4);
               SendPkt(fileNameRedirector,pktTCP);
               auxInfo->rstRecibido = 1;
               auxInfo->timerID = timeSetEvent((TIME_OVER_SC+1)*2, 0,IntRST, (unsigned long)auxInfo,
TIME_ONESHOT);
            }
         }
         return 0;
      }
      pktTCP->usPacActual   = 1;
      pktTCP->usNumPacTotal = 1;

      // Creamos el nodo nuevo en el que se encontrará el  valor a añadir
      newHash           = (hashTable*)calloc(1,sizeof(hashTable));
      newHash->ipSource  = pktTCP->ullPSource;
      newHash->ptoSource = pktTCP->usPortSourceTCP;
      newHash->next      = NULL;

      //Almacenamos la información de la conexión
      newInfo                = (infoHash*)calloc(1,sizeof(infoHash));
      newInfo->ipDest         = pktTCP->ullPDestination;
      newInfo->ptoDest        = pktTCP->usPortDestinationTCP;
      newInfo->totalPktCap    = 1;
```

```
    newInfo->totalPktAda     = 0;
    newInfo->endConection    = (unsigned char)0;
    newInfo->rstRecibido     = 0;
    newInfo->numAsentimiento = (unsigned long)rand(); // El discriminador elige uno aleatorio
    newInfo->numSecPkt       = pktTCP->ulSecuenceNumber + 1 ; //nº secuencia paquete esperado
    newInfo->congestion      = 0;
    newInfo->next            = NULL;
    newInfo->reintentosScSc  = 0;
    newInfo->tamWin          = pktTCP->usWin;

    // Almacenamos el paquete en el buffer
    newInfo->buffer         = (bufferPkt*)calloc(1,sizeof(bufferPkt));
    newInfo->buffer->pkt    = *pktTCP;
    newInfo->buffer->prev   = NULL;
    newInfo->buffer->next   = NULL;
    newInfo->conexDual      = NULL;
    newInfo->hiloRedirector = NULL;
    newInfo->hiloCapturador = NULL;
    newHash->info           = newInfo;
    gHashCapturador         = newHash;

    //Es un SYN "puro" sin ACK para establecer la conexion
    if (pktTCP->ucFack == (unsigned char)0){
    //Si Fsin =1, siempre va a entrar aki, ya que antes hemos comprobado que Fsinc = 1 and Fack = 1
       printf("Enviando SYN-ACK\n");
       CreateSendSynAck(pktTCP,newInfo->numAsentimiento,1,1,true,fileNameEnv);
       newInfo->numAsentimiento++;
       newInfo->timerID = timeSetEvent(TIMER_SYN, 0,IntSYNACK, (unsigned long)newInfo, TIME_ONESHOT);
    }
    return 0;
}

// LA HASH YA EXISTE: Buscamos si tiene una entrada para esa IP-Pto Origen
for (auxHash = gHashCapturador; auxHash != NULL ; auxHash = auxHash->next){
    /* Existe una entrada en la hash para esa conexión Origen*/
    if ((auxHash->ipSource == pktTCP->ulIPSource)  && (auxHash->ptoSource == pktTCP->usPortSourceTCP)){
       /* Buscamos si existe una entrada para ese destino*/
       for (auxInfo = auxHash->info; auxInfo != NULL ;auxInfo = auxInfo->next){
          /* Guardamos el último elemento*/
          if (auxInfo->next == NULL){
             endInfo = auxInfo;
          }
          /* Existe una entrada en la hash para esas conexiones*/
          if ((auxInfo->ipDest == pktTCP->ulIPDestination)  && (auxInfo->ptoDest == pktTCP->usPortDestinationTCP)){
             if (auxInfo->endConection == (unsigned char) 2){
                printf("(CheckPktTCPCapturador)Conexión cerrada en los dos extremos, Pkt ignorado\n");
                //Ya he cerrado la conexion, pero estoy esperando un tiempo para pkt perdidos
                //este pkt no es un ack ni un RST
                return 0;
             }
             timeKillEvent(auxInfo->timerID);

             //Existe la entrada en la hash pero es un SYN (duplicado)
             if (pktTCP->ucFsinc == (unsigned char)1 ){
                printf("(CheckPktTCPCapturador)SYN duplicado\n");
                return 0;
             }
             //Comprobamos si ya habiamos recibido un RST
             if (auxInfo->rstRecibido == 1) {
                printf("(CheckPktTCPCapturador)Ya habiamos recibido un RST\n");
                return 0;
             }
             //Analizamos si el paquete es un RESET
             if (pktTCP->ucFrstConnect == (unsigned char)1){
                printf("(CheckPktTCPCapturador)RST recibido\n");
                //Enviar el RST al redirector con campos de reset recibido a 1
                auxInfo->rstRecibido = 1;
                if (auxInfo->conexDual != NULL) {
                   timeKillEvent(auxInfo->timerScSc);
                   //si habia algun hilo lanzado lo matamos
                   if (auxInfo->hiloCapturador != NULL) {
                      TerminateThread(auxInfo->hiloCapturador,0);
                      auxInfo->hiloCapturador = NULL;
                   }
                   if (auxInfo->hiloRedirector != NULL) {
```

```
                    TerminateThread(auxInfo->hiloRedirector,0);
                    auxInfo->hiloRedirector = NULL;
                }
                auxInfo->conexDual->rstRecibido = 1;
                if (auxInfo->conexDual->hiloCapturador != NULL) {
                    TerminateThread(auxInfo->conexDual->hiloCapturador,0);
                    auxInfo->conexDual->hiloCapturador = NULL;
                }
                if (auxInfo->conexDual->hiloRedirector != NULL) {
                    TerminateThread(auxInfo->conexDual->hiloRedirector,0);
                    auxInfo->conexDual->hiloRedirector = NULL;
                }
                printf("(CheckPktTCPCapturador)Enviando RST al redirecftor\n");
                CreateSendReset (pktTCP, 0, fileNameRedirector);
                MyStat->pktSndRed++;
                MyStat->bytSndRed = MyStat->bytSndRed + pktTCP->uiLengthData - (pktTCP->ucDataOffset * 4);
                MyStat->pktSndRedTCP++;
                MyStat->bytSndRedTCP = MyStat->bytSndRedTCP + pktTCP->uiLengthData - (pktTCP-
>ucDataOffset * 4);

                auxInfo->reintentosScSc = 0;
                if (auxInfo->buffer == NULL) {
                    auxBuffer = (bufferPkt*)calloc(1,sizeof(bufferPkt));
                    auxBuffer->pkt  = *pktTCP;
                    auxBuffer->prev = NULL;
                    auxBuffer->next = NULL;
                    auxInfo->buffer = auxBuffer;
                }
                auxInfo->timerID = timeSetEvent((TIME_OVER_SC+1)*2, 0,IntRST, (unsigned long)auxInfo,
TIME_ONESHOT);
            }
            else {
                //No existe la conexion dual, RST a un SYN-ACK
                DeleteConectionCapturador(pktTCP->ulIPSource, pktTCP->usPortSourceTCP, pktTCP-
>ulIPDestination, pktTCP->usPortDestinationTCP);
            }
            return 0;
        }
        if (auxInfo->congestion == 1) {
            printf("(CheckPktTCPCapturador)Teniamos congestion Activa\n");
            CreateSendAckWin0(pktTCP,auxInfo->numAsentimiento, auxInfo->numSecPkt, 0, 1);
            return 0;
        }
        if (auxInfo->endConection  == (unsigned char) 1) {
            printf("(CheckPktTCPCapturador)Ya habiamos recibido un FIN\n");
            //timeKillEvent(auxInfo->timerID);
            if (pktTCP->ucFdataEnd == (unsigned char) 1) {
                CreateSendAckWin0(pktTCP,auxInfo->numAsentimiento, auxInfo->numSecPkt, 0, 1);
            } else {
                return -1;
            }
            return 0;
        }
        // Vemos si hay más paquetes en el buffer
        printf("(CheckPktTCPCapturador)ordenando pkt en el buffer\n");
        bufferOrdenado  = MergeBufferCapturador (auxInfo,pktTCP);
        auxInfo->buffer = bufferOrdenado;

        // Recepción de paquete FIN: Mandamos todos los paquete del buffer al redirector
        if (auxInfo->endConection == (unsigned char) 1){
            // Tras recibir el FIN: guardamos el totalPktCap en
            // totalPktAda y ponemos totalPktCap a 0
            printf("(CheckPktTCPCapturador)FIN recibido, enviando al Redirector\n");
            auxInfo->totalPktAda = auxInfo->totalPktCap;
            auxInfo->totalPktCap = 0;
            auxInfo->hiloRedirector =
CreateThread(NULL,0,(LPTHREAD_START_ROUTINE)SendBuffer2Redirector,auxInfo,0,&thId_Red);
            //Lanzar Timer com SC-SC
            auxInfo->timerScSc = timeSetEvent(TIME_OVER_SC*(auxInfo->totalPktAda + 1),
0,int_SC_SC_Capturador, (unsigned long)auxInfo, TIME_ONESHOT);
            return 0;
        }//if FIN = 1
        // Si tenemos almacenado en el buffer el máximo nº posible
        // de paquetes:
        //          - Mandamos ACK con WIN=0
```

```
                //            - Mandamos el buffer al redirector
                if (auxInfo->totalPktCap >= MAX_BUFFER){
                    printf("(CheckPktTCPCapturador)Congestion detectada...Enviando al redirector\n");
                    auxInfo->congestion  = 1;
                    auxInfo->totalPktAda = auxInfo->totalPktCap;
                    auxInfo->totalPktCap = 0;
                    CreateSendAckWin0(pktTCP,auxInfo->numAsentimiento, auxInfo->numSecPkt, 0, 1);
                    auxInfo->hiloRedirector =
CreateThread(NULL,0,(LPTHREAD_START_ROUTINE)SendBuffer2Redirector,auxInfo,0,&thId_Red);
                    //Lanzar Timer com SC-SC
                    auxInfo->timerScSc = timeSetEvent(TIME_OVER_SC*(auxInfo->totalPktAda + 1),
0,int_SC_SC_Capturador, (unsigned long)auxInfo, TIME_ONESHOT);
                    return 0;
                }
                //Vuelvo a lanzar el timer si no es un fin
                auxInfo->timerID = timeSetEvent(TIME_OVER, 0,IntData, (unsigned long)auxInfo, TIME_ONESHOT);
                return 0;
            }//if existe destino
        }//for infoHash
        // No existe una conexión destino para esa IP
        // Analizamos si el paquete es un SYN
        if ((pktTCP->ucFsinc != (unsigned char)1)){
            if (pktTCP->ucFrstConnect != 1) {
                //mandar un reset
                CreateSendReset (pktTCP, 1, fileNameEnv);
            } else {
                //Buscar en la hash del adaptador y si existe, borrar conexion
                auxInfo = searchReverseConnection(pktTCP, gHashAdaptador);
                if (auxInfo != NULL) {
                    //Estamos ante un rst como respuesta a un syn -> mandar reset al otro extremo
                    printf("(CheckPktTCPCapturador)Recibido RST como respuesta a un SYN, enviando RST al
redirector\n");
                    timeKillEvent(auxInfo->timerID);
                    MyStat->pktSndRed++;
                    MyStat->bytSndRed = MyStat->bytSndRed + pktTCP->uiLengthData - (pktTCP->ucDataOffset * 4);
                    MyStat->pktSndRedTCP++;
                    MyStat->bytSndRedTCP = MyStat->bytSndRedTCP + pktTCP->uiLengthData - (pktTCP->ucDataOffset *
4);
                    SendPkt(fileNameRedirector,pktTCP);
                    auxInfo->rstRecibido = 1;
                    auxInfo->timerID = timeSetEvent((TIME_OVER_SC+1)*2, 0,IntRST, (unsigned long)auxInfo,
TIME_ONESHOT);
                }
            }
            return 0;
        }
        pktTCP->usPacActual     = 1;
        pktTCP->usNumPacTotal   = 1;
        newInfo                 = (infoHash*)calloc(1,sizeof(infoHash));
        newInfo->ipDest         = pktTCP->ulIPDestination;
        newInfo->ptoDest        = pktTCP->usPortDestinationTCP;
        newInfo->totalPktCap    = 1;
        newInfo->totalPktAda    = 0;
        newInfo->endConection   = (unsigned char)0;
        newInfo->rstRecibido    = 0;
        newInfo->numAsentimiento = (unsigned long)rand(); // El discriminador elige uno aleatorio
        newInfo->numSecPkt      = pktTCP->ulSecuenceNumber + 1 ; //nº secuencia paquete esperado
        newInfo->congestion     = 0;
        newInfo->buffer         = (bufferPkt*)calloc(1,sizeof(bufferPkt));
        newInfo->buffer->pkt    = *pktTCP;
        newInfo->buffer->prev   = NULL;
        newInfo->buffer->next   = NULL;
        newInfo->next           = NULL;
        newInfo->conexDual      = NULL;
        newInfo->reintentosScSc  = 0;
        newInfo->tamWin         = pktTCP->usWin;
        newInfo->hiloRedirector  = NULL;
        newInfo->hiloCapturador  = NULL;

        //Añadimos la nueva conexión destino al final.
        endInfo->next = newInfo;
        if (pktTCP->ucFack == (unsigned char)0){
            CreateSendSynAck(pktTCP,newInfo->numAsentimiento,1,1,true,fileNameEnv);
            newInfo->timerID = timeSetEvent(TIMER_SYN, 0,IntSYNACK, (unsigned long)newInfo, TIME_ONESHOT);
            newInfo->numAsentimiento++;
```

```
            }
            return 0;
        }// if origen =
    } //fin for ghashCapturador

    // Si llegamos aquí es porque no existe una entrada en la hash con esa conexión
    // Creamos el nodo nuevo en el que se encontrará el  valor a añadir
    if ((pktTCP->ucFsinc != (unsigned char)1)){
        if (pktTCP->ucFrstConnect != 1) {
            //mandar un reset
            CreateSendReset (pktTCP, 1, fileNameEnv);
        } else {
            //Buscar en la hash del adaptador y si existe, borrar conexion
            auxInfo = searchReverseConnection(pktTCP, gHashAdaptador);
            if (auxInfo != NULL) {
                //Estamos ante un rst como respuesta a un syn -> mandar reset al otro extremo
                printf("(CheckPktTCPCapturador)Recibido RST como respuesta a un SYN, enviando RST al redirector\n");
                timeKillEvent(auxInfo->timerID);
                MyStat->pktSndRed++;
                MyStat->bytSndRed = MyStat->bytSndRed + pktTCP->uiLengthData - (pktTCP->ucDataOffset * 4);
                MyStat->pktSndRedTCP++;
                MyStat->bytSndRedTCP = MyStat->bytSndRedTCP + pktTCP->uiLengthData - (pktTCP->ucDataOffset * 4);
                SendPkt(fileNameRedirector,pktTCP);
                auxInfo->rstRecibido = 1;
                auxInfo->timerID = timeSetEvent((TIME_OVER_SC+1)*2, 0,IntRST, (unsigned long)auxInfo,
TIME_ONESHOT);
            }
        }
        return 0;
    }

    //El paquete es un SYN, creamos entrada en la hash
    newHash           = (hashTable*)calloc(1,sizeof(hashTable));
    newHash->ipSource  = pktTCP->ulIPSource;
    newHash->ptoSource = pktTCP->usPortSourceTCP;
    newHash->next      = gHashCapturador;
    gHashCapturador    = newHash;

    pktTCP->usPacActual   = 1;
    pktTCP->usNumPacTotal = 1;
    newInfo             = (infoHash*)calloc(1,sizeof(infoHash));
    newInfo->ipDest       = pktTCP->ulIPDestination;
    newInfo->ptoDest      = pktTCP->usPortDestinationTCP;
    newInfo->totalPktCap  = 1;
    newInfo->totalPktAda  = 0;
    newInfo->endConection = (unsigned char)0;
    newInfo->rstRecibido  = 0;

    newInfo->numAsentimiento = (unsigned long)rand();   // El discriminador elige uno aleatorio
    newInfo->numSecPkt       = pktTCP->ulSecuenceNumber + 1 ; // nº secuencia paquete esperado
    newInfo->congestion      = 0;
    newInfo->reintentosScSc  = 0;
    newInfo->tamWin          = pktTCP->usWin;
    newInfo->buffer          = (bufferPkt*)calloc(1,sizeof(bufferPkt));
    newInfo->buffer->pkt     = *pktTCP;
    newInfo->buffer->prev    = NULL;
    newInfo->buffer->next    = NULL;
    newInfo->conexDual       = NULL;
    newInfo->hiloRedirector  = NULL;
    newInfo->hiloCapturador  = NULL;
    newHash->info            = newInfo;

    //Es un SYN "puro" sin ACK para establecer la conexion
    if (pktTCP->ucFack == (unsigned char)0){
        CreateSendSynAck(pktTCP,newInfo->numAsentimiento,1,1,true,fileNameEnv);
        newInfo->timerID = timeSetEvent(TIMER_SYN, 0,IntSYNACK, (unsigned long)newInfo, TIME_ONESHOT);
        newInfo->numAsentimiento++;
    }
    return 0;
}//CheckPktTCPCapturador


/*************************************************************************************************************
NOMBRE:      CheckPktTCPAdaptador
SINOPSIS:    Recibe un paquete del Adaptador y lo analiza en funcion de la
```

```
            optimizacion del protocolo TCP.
PARAMETROS:  HeaderPacket * (entrada): pkt Recibido
DESCRIPCION: Para comunicaciones Simetricas (Sc-Sc). Comprueba que tipo de
            paquete ha recibido: Ack-Sc-Sc, RST, Ack-Rst o pkt TCP. Si es un
            Ack-Sc-Sc significa que el otro extremo nos ha confirmados los
            paquetes enviados, que no va a pedir Rtx y por lo tanto podemos
            liberar el buffer. Ack-Rst indica que el otro Sc ha recibido nuestro
            pkt de RST y podemos borrar la conexion. Si es un RST debemos
            comunicarle al otro SC que lo hemos recibido , enviarlo al
            Capturador y liberar la conexion. Si es un pkt TCP, debemos esperar
            ha recibir todos los paquetes que indica el campo usNumPacTotal,
            ordenarlos y enviarlos al Capturador.
*****************************************************************************************************************/
int CheckPktTCPAdaptador(Fifo_Adapter *elementoFifo){

   hashTable *auxHash,*newHash;
   infoHash  *auxInfo, *infoRtx, *endInfo, *newInfo;
   bufferPkt *auxBuffer, *bufferOrdenado;
   DWORD thId_Adap = 0;
   int conexionCerrada;
   unsigned long IpOrigen, IpDestino;
   unsigned short portOrigen, portDestino;
   HeaderPacket *pktTCP;


   printf("(CheckPktTCPAdaptador) paquete recibido\n");
   pktTCP = &(elementoFifo->pkt);
   if (pktTCP->ucFrstConnect == (unsigned char)1) {
      printf("RST recibido\n");
      //Send ack del rst al otro Sc
      CreateSendAckReset(pktTCP);
      CreateSendAckReset(pktTCP);
      CreateSendAckReset(pktTCP);

      infoRtx = searchDirectConnection(pktTCP,gHashAdaptador);
      if (infoRtx != NULL) {
         auxInfo = infoRtx->conexDual;
         timeKillEvent(infoRtx->timerID);
         timeKillEvent(infoRtx->timerScSc);
         IpOrigen    = pktTCP->ulIPSource;
         IpDestino   = pktTCP->ulIPDestination;
         portOrigen  = pktTCP->usPortSourceTCP;
         portDestino = pktTCP->usPortDestinationTCP;

         DeleteConectionAdaptador(IpOrigen, portOrigen, IpDestino, portDestino);
         if (auxInfo != NULL) {
            //Send reset al capturador
            pktTCP->ulAckNumber      = auxInfo->numSecPkt;
            CreateSendResetDual(pktTCP,auxInfo, fileNameEnv);
            timeKillEvent(auxInfo->timerID);
            timeKillEvent(auxInfo->timerScSc);
            DeleteConectionCapturador (IpDestino,portDestino,IpOrigen,portOrigen);
         }
      }
      else {
         //Realizar busqueda en la otra hash por si acaso
         infoRtx = searchReverseConnection(pktTCP,gHashCapturador);
         if (infoRtx != NULL) {
            //Send reset al capturador
            pktTCP->ulSecuenceNumber = infoRtx->numAsentimiento;
            CreateSendReset (pktTCP, 0, fileNameEnv);
            timeKillEvent(infoRtx->timerID);
            timeKillEvent(infoRtx->timerScSc);
            DeleteConectionCapturador (pktTCP->ulIPDestination,pktTCP->usPortDestinationTCP,pktTCP-
>ulIPSource,pktTCP->usPortSourceTCP);
         }
      }
    return 0;
   }

   if ((pktTCP->usNumPacTotal == 0) && (pktTCP->usPacActual == 0)) {
      //Código no valido
      printf("Codigo no valido\n");
      return -1;
   }
```

```
   //Comprobar si el pkt es un ACK de la comunicacion SC_SC
   //ACK de comunicacion SC-SC:
   //                    - usNumPacTotal = 0
   //                    - usPacActual   = n
   if (pktTCP->usNumPacTotal == 0 && pktTCP->usPacActual != 0){
      printf("Comunicacion SC-SC: ");
      //Comunicacion SC_SC
      //Buscar en la hash del Capturador esa conexion
      infoRtx = searchReverseConnection(pktTCP, gHashCapturador);
      if (infoRtx != NULL) {
         if (infoRtx->rstRecibido == 1){
            printf("RST activo\n");
            return 0;
         }
         if (infoRtx->endConection == (unsigned char) 2) {
            printf("Cierre de conexion activo\n");
            return 0;
         }
         //Comprobar si es ACK o NACK
         if (pktTCP->usPacActual > infoRtx->totalPktAda) {
            printf("Confirmados todos los pkt\n");
            //matar el timer SC-SC
            timeKillEvent(infoRtx->timerScSc);
            infoRtx->reintentosScSc = 0;
            //Todos los paquetes se han recibidos y han sido asentidos por
            //el otro extremo:
            //            - Borramos el buffer
            //            - Si existía congestion: mandamos WIN updade
            conexionCerrada = DeleteBuffer(infoRtx,1);
            if (conexionCerrada == 0) {
               if (infoRtx->congestion == 1){
                  infoRtx->congestion = 0;
                  CreateSendAckWin0(pktTCP,infoRtx->numAsentimiento,infoRtx->numSecPkt, 4*MSS,0);
               }
               infoRtx->totalPktAda = 0;
            }
         }
         else {
            printf("Rtx solicitada\n");
            //Reiniciamos el timer timer SC-SC
            timeKillEvent(infoRtx->timerScSc);
            //Rtx paquete cuyo usPacActual coincida
            RtxPkt(infoRtx,pktTCP->usPacActual);
            infoRtx->reintentosScSc = 0;
            infoRtx->timerScSc = timeSetEvent(TIME_OVER_SC*(pktTCP->usNumPacTotal + 1),
0,int_SC_SC_Capturador, (unsigned long)infoRtx, TIME_ONESHOT);
         }
      }
      return 0;
   }

   if (pktTCP->usNumPacTotal == 1 && pktTCP->usPacActual == 0) {
      //ACK de un rst Sc-Sc
      printf("ACK de un RST\n");
      infoRtx = searchReverseConnection(pktTCP,gHashCapturador);
      if (infoRtx != NULL) {
            printf("Conexion en el capturador\n");
         auxInfo = infoRtx->conexDual;
         timeKillEvent(infoRtx->timerID);
         timeKillEvent(infoRtx->timerScSc);
         DeleteConectionCapturador (pktTCP->ulIPDestination, pktTCP->usPortDestinationTCP,pktTCP->ulIPSource,
pktTCP->usPortSourceTCP);
         if (auxInfo != NULL) {
            timeKillEvent(auxInfo->timerID);
            timeKillEvent(auxInfo->timerScSc);
            DeleteConectionAdaptador(pktTCP->ulIPSource, pktTCP->usPortSourceTCP, pktTCP->ulIPDestination,
pktTCP->usPortDestinationTCP);
         }
      }
      else {
         //buscar por la otra ghash, por si no se ha podido establecer la conexion
         infoRtx = searchDirectConnection(pktTCP,gHashAdaptador);
         if (infoRtx != NULL) {
            printf("Conexion en el adaptador\n");
```

```
            timeKillEvent(infoRtx->timerID);
            timeKillEvent(infoRtx->timerScSc);
            DeleteConectionAdaptador(pktTCP->ulIPSource, pktTCP->usPortSourceTCP, pktTCP->ulIPDestination,
pktTCP->usPortDestinationTCP);
         }
      }
      return 0;
   }

   // La hash no existe
   if (gHashAdaptador == NULL){
      printf("hash nula\n");
      //Creamos la tabla
      newHash           = (hashTable*)calloc(1,sizeof(hashTable));
      newHash->ipSource  = pktTCP->ulIPSource;
      newHash->ptoSource = pktTCP->usPortSourceTCP;
      newHash->next      = NULL;

      //Almacenamos la información de la conexión
      newInfo          = (infoHash*)calloc(1,sizeof(infoHash));
      newInfo->ipDest  = pktTCP->ulIPDestination;
      newInfo->ptoDest = pktTCP->usPortDestinationTCP;

      newInfo->totalPktAda    = pktTCP->usNumPacTotal;
      newInfo->totalPktAda--; //decrementamos porque hemos recibido uno
      printf("Faltan %d pkt\n", newInfo->totalPktAda);
      newInfo->totalPktCap    = 0;
      newInfo->endConection   = (unsigned char) 0;
      newInfo->rstRecibido    = 0;
      newInfo->reintentosScSc = 0;
      newInfo->tamWin         = pktTCP->usWin;
      newInfo->hiloRedirector = NULL;
      newInfo->hiloCapturador = NULL;
      if (elementoFifo->timer > 0) {
         newInfo->vCanal = elementoFifo->timer;
      } else {
         newInfo->vCanal = TIMER_LENTO;
      }
      //printf("Timer: %d\n",newInfo->vCanal);

      if (pktTCP->ucFsinc == 1) {
         newInfo->numAsentimiento = pktTCP->ulSecuenceNumber;
         newInfo->numSecPkt       = pktTCP->ulSecuenceNumber + 1;
      }
      newInfo->next = NULL;

      //Paquetes intercambiados en esa conexión
      newInfo->buffer       = (bufferPkt*)calloc(1,sizeof(bufferPkt));
      newInfo->buffer->pkt  = *pktTCP;
      newInfo->buffer->prev = NULL;
      newInfo->buffer->next = NULL;
      newInfo->conexDual    = NULL;

      //Lanzar el Timer SC_SC adaptador
      //newInfo->timerScSc = timeSetEvent(newInfo->vCanal, 0,int_SC_SC_Adaptador, (unsigned long)newInfo,
TIME_ONESHOT);

      newHash->info        = newInfo;
      gHashAdaptador       = newHash;

      if (newInfo->totalPktAda == 0){
         newInfo->totalPktCap = pktTCP->usNumPacTotal;
         if (newInfo->buffer->pkt.ucFsinc == (unsigned char)1){
            printf("Enviando SYN\n");
            //MANDO SOLO EL SYN Y CUANDO ME LLEGUE EL SYN-ACK LOS DATOS
            CreateSendSynAck(&(newInfo->buffer->pkt),newInfo->buffer-
>pkt.ulSecuenceNumber,1,0,false,fileNameEnv);
            //Lanzar un timer para ver si recibo contestacion al SYN
            newInfo->timerID = timeSetEvent(TIMER_SYN, 0,lntSYN, (unsigned long)newInfo, TIME_ONESHOT);
         }
         else {
            printf("Lanzando hilo dual\n");
            newInfo->hiloCapturador =
CreateThread(NULL,0,(LPTHREAD_START_ROUTINE)SendBuffer2CapturadorDual,newInfo,0,&thId_Adap);
//Segundo Hilo
```

```
        }
      }else {
         newInfo->timerScSc = timeSetEvent(newInfo->vCanal, 0,int_SC_SC_Adaptador, (unsigned long)newInfo,
TIME_ONESHOT);
      }

      return 0;
   }//if gHashAdaptador = NULL

   // La hash del adaptador EXISTE: buscamos si ya existe una entrada para esa IP-Pto Origen
   for (auxHash = gHashAdaptador; auxHash != NULL; auxHash = auxHash->next){
      // Existe una entrada en la hash para esa conexión Origen
      if ((auxHash->ipSource == pktTCP->ulIPSource)  && (auxHash->ptoSource == pktTCP->usPortSourceTCP)){
         // Buscamos si existe una entrada para ese destino
         for (auxInfo = auxHash->info; auxInfo != NULL ;auxInfo = auxInfo->next){
            // Guardamos el último elemento
            if (auxInfo->next == NULL){
               endInfo = auxInfo;
            }
            // Existe una entrada en la hash para esas conexiones
            if ((auxInfo->ipDest == pktTCP->ulIPDestination)  && (auxInfo->ptoDest == pktTCP->usPortDestinationTCP)){
               // Si ya hemos detectamos un RST para esa conexion, obviamos los datos
               timeKillEvent(auxInfo->timerScSc);
               //actualizamos la velocidad del canal
               if (elementoFifo->timer > 0) {
                  auxInfo->vCanal = elementoFifo->timer;
               } else {
                  auxInfo->vCanal = TIMER_LENTO;
               }
               if (auxInfo->rstRecibido == 1){
                  return 0;
               }
               if (auxInfo->endConection == (unsigned char) 2) {
                  return 0;
               }
               auxInfo->reintentosScSc = 0;
               if (pktTCP->ucFsinc == 1) {
                  auxInfo->numAsentimiento = pktTCP->ulSecuenceNumber;
                  auxInfo->numSecPkt       = pktTCP->ulSecuenceNumber + 1;
               }
               // Buffer con paquetes
               if (auxInfo->buffer == NULL){
                  auxInfo->totalPktAda = pktTCP->usNumPacTotal;//nº pkt a recibir
               }
               printf("Ordenando buffer\n");
               bufferOrdenado = MergeBufferAdaptador (auxInfo,pktTCP);
               auxInfo->buffer = bufferOrdenado;
               if(bufferOrdenado == NULL){
                  printf("Buffer nulo\n");
                  return 0;
               }
               printf("Faltan %d paquetes\n", auxInfo->totalPktAda);
               if (auxInfo->totalPktAda == 0){
                  if (auxInfo->totalPktCap != 0) {
                     printf("Ya estabamos enviando al Capturador\n");
                     return 0;
                  }
                  //Matar timer
                  timeKillEvent(auxInfo->timerScSc);

                  auxInfo->totalPktCap = pktTCP->usNumPacTotal;
                  if (auxInfo->buffer->pkt.ucFsinc == (unsigned char)1){
                     //MANDO SOLO EL SYN Y CUANDO ME LLEGUE EL SYN-ACK LOS DATOS
                     printf("Enviando SYN\n");
                     CreateSendSynAck(&(auxInfo->buffer->pkt),auxInfo->buffer-
>pkt.ulSecuenceNumber,1,0,false,fileNameEnv);
                     //Lanzar un timer para ver si recibo contestacion al SYN
                     auxInfo->timerID = timeSetEvent(TIMER_SYN, 0,IntSYN, (unsigned long)auxInfo, TIME_ONESHOT);
                  }
                  else {
                     printf("Lanzando hilo dual\n");
                     auxInfo->hiloCapturador =
CreateThread(NULL,0,(LPTHREAD_START_ROUTINE)SendBuffer2CapturadorDual,auxInfo,0,&thId_Adap); //Segundo
Hilo
                  }
```

```
                }else {
                    auxInfo->timerScSc = timeSetEvent(auxInfo->vCanal, 0,int_SC_SC_Adaptador, (unsigned long)auxInfo,
TIME_ONESHOT);
                }
                return 0;
              } //fi destino =
            }//for infoHash
            // No existe una conexión destino para esa IP origen.
            newInfo         = (infoHash*)calloc(1,sizeof(infoHash));
            newInfo->ipDest  = pktTCP->ulIPDestination;
            newInfo->ptoDest = pktTCP->usPortDestinationTCP;

            newInfo->totalPktAda  = pktTCP->usNumPacTotal;
            newInfo->totalPktAda--;
            newInfo->totalPktCap  = 0;
            newInfo->endConection = (unsigned char)0;
            newInfo->rstRecibido  = 0;
            if (pktTCP->ucFsinc == 1) {
               newInfo->numAsentimiento = pktTCP->ulSecuenceNumber;
               newInfo->numSecPkt       = pktTCP->ulSecuenceNumber + 1;
            }
            newInfo->buffer = (bufferPkt*)calloc(1,sizeof(bufferPkt));
            newInfo->buffer->pkt    = *pktTCP;
            newInfo->buffer->prev   = NULL;
            newInfo->buffer->next   = NULL;
            newInfo->conexDual      = NULL;
            newInfo->next           = NULL;
            newInfo->reintentosScSc = 0;
            newInfo->tamWin         = pktTCP->usWin;
            newInfo->hiloRedirector = NULL;
            newInfo->hiloCapturador = NULL;
            if (elementoFifo->timer > 0) {
              newInfo->vCanal = elementoFifo->timer;
            } else {
              newInfo->vCanal = TIMER_LENTO;
            }

            //Lanzar Timer SC_SC adaptador
            //newInfo->timerScSc = timeSetEvent(newInfo->vCanal, 0,int_SC_SC_Adaptador, (unsigned long)newInfo,
TIME_ONESHOT);

            //Añadimos la nueva conexión destino al final.
            endInfo->next = newInfo;
            if (newInfo->totalPktAda == 0){
               newInfo->totalPktCap = pktTCP->usNumPacTotal;
               if (newInfo->buffer->pkt.ucFsinc == (unsigned char)1){
                  //MANDO SOLO EL SYN Y CUANDO ME LLEGUE EL SYN-ACK LOS DATOS
                  printf("Enviando SYN\n");
                  CreateSendSynAck(&(newInfo->buffer->pkt),newInfo->buffer-
>pkt.ulSecuenceNumber,1,0,false,fileNameEnv);
                  //Lanzar un timer para ver si recibo contestacion al SYN
                  newInfo->timerID = timeSetEvent(TIMER_SYN, 0,IntSYN, (unsigned long)newInfo, TIME_ONESHOT);
               }
               else {
                  printf("Lanzando hilo dual\n");
                  newInfo->hiloCapturador =
CreateThread(NULL,0,(LPTHREAD_START_ROUTINE)SendBuffer2CapturadorDual,newInfo,0,&thId_Adap);
//Segundo Hilo
               }
            }else {
               newInfo->timerScSc = timeSetEvent(newInfo->vCanal, 0,int_SC_SC_Adaptador, (unsigned long)newInfo,
TIME_ONESHOT);
            }


            return 0;
          }//if origen =
        } //fin for ghashAdaptador

        // Si llegamos aquí es porque no existe una entrada en la hash con esa conexión
        // Creamos el nodo nuevo en el que se encontrará el  valor a añadir
        newHash           = (hashTable*)calloc(1,sizeof(hashTable));
        newHash->ipSource  = pktTCP->ulIPSource;
        newHash->ptoSource = pktTCP->usPortSourceTCP;
        newHash->next      = gHashAdaptador;
```

```
   gHashAdaptador     = newHash;

   newInfo           = (infoHash*)calloc(1,sizeof(infoHash));
   newInfo->ipDest    = pktTCP->ulIPDestination;
   newInfo->ptoDest   = pktTCP->usPortDestinationTCP;
   newInfo->conexDual = NULL;
   newInfo->totalPktAda = pktTCP->usNumPacTotal;
   newInfo->totalPktCap = 0;
   newInfo->totalPktAda--;
   newInfo->endConection = (unsigned char)0;
   newInfo->rstRecibido  = 0;
   newInfo->tamWin       = pktTCP->usWin;
   newInfo->hiloRedirector = NULL;
   newInfo->hiloCapturador = NULL;
   if (elementoFifo->timer > 0) {
      newInfo->vCanal = elementoFifo->timer;
   } else {
      newInfo->vCanal = TIMER_LENTO;
   }

   if (pktTCP->ucFsinc == 1) {
      newInfo->numAsentimiento = pktTCP->ulSecuenceNumber;
      newInfo->numSecPkt       = pktTCP->ulSecuenceNumber + 1;
   }

   newInfo->buffer       = (bufferPkt*)calloc(1,sizeof(bufferPkt));
   newInfo->buffer->pkt  = *pktTCP;
   newInfo->buffer->prev = NULL;
   newInfo->buffer->next = NULL;
   newInfo->reintentosScSc  = 0;
   newHash->info = newInfo;

   //Lanzar Timer SC_SC adaptador
   //newInfo->timerScSc = timeSetEvent(newInfo->vCanal, 0,int_SC_SC_Adaptador, (unsigned long)newInfo,
TIME_ONESHOT);
   if (newInfo->totalPktAda == 0){
      newInfo->totalPktCap = pktTCP->usNumPacTotal;
      if (newInfo->buffer->pkt.ucFsinc == (unsigned char)1){
         //MANDO SOLO EL SYN Y CUANDO ME LLEGUE EL SYN-ACK LOS DATOS
         printf("Enviando SYN\n");
         CreateSendSynAck(&(newInfo->buffer->pkt),newInfo->buffer->pkt.ulSecuenceNumber,1,0,false,fileNameEnv);
         //Lanzar un timer para ver si recibo contestacion al SYN
         newInfo->timerID = timeSetEvent(TIMER_SYN, 0,IntSYN, (unsigned long)newInfo, TIME_ONESHOT);
      }
      else {
         printf("Lanzando hilo dual\n");
         newInfo->hiloCapturador =
CreateThread(NULL,0,(LPTHREAD_START_ROUTINE)SendBuffer2CapturadorDual,newInfo,0,&thId_Adap);
//Segundo Hilo
      }
   }else {
      newInfo->timerScSc = timeSetEvent(newInfo->vCanal, 0,int_SC_SC_Adaptador, (unsigned long)newInfo,
TIME_ONESHOT);
   }


   return 0;
}//CheckPktTCPAdaptador



/*************************************************************************************************
NOMBRE:      NewConectionCapturador
SINOPSIS:    crea una nueva conexion en la hash del capturador
PARAMETROS:  HeaderPacket * (entrada): pkt con la informacion de la conexion
             infoHash * (entrada): elemento Dual de la hash del Adaptador
DESCRIPCION: Crea una nueva conexión en la hash del Capturador con la información
             del pkt recibido y se enlaza con su conexion Dual de la hash del
             Adaptador
*************************************************************************************************/
void NewConectionCapturador(HeaderPacket *pktTCP, infoHash *infoHashAdap){
   hashTable *newHash, *auxHash;
   infoHash  *newInfo;
   int encontrado;
```

```
   //Almacenamos la información de la conexión
   newInfo                = (infoHash*)calloc(1,sizeof(infoHash));
   newInfo->ipDest        = pktTCP->ulIPDestination;
   newInfo->ptoDest       = pktTCP->usPortDestinationTCP;
   newInfo->endConection  = (unsigned char) 0;
   newInfo->rstRecibido   = 0;
   newInfo->numAsentimiento = pktTCP->ulAckNumber;
   newInfo->numSecPkt     = pktTCP->ulSecuenceNumber + 1 ; //nº secuencia paquete esperado
   newInfo->next          = NULL;
   newInfo->conexDual     = infoHashAdap;//enlazamos con la hash dual
   newInfo->tamWin        = pktTCP->usWin;
   newInfo->reintentosScSc  = 0;
   newInfo->hiloRedirector  = NULL;
   newInfo->hiloCapturador  = NULL;
   infoHashAdap->conexDual  = newInfo;
   //newInfo->vCanal        = TIMER_LENTO;

   //Añadimos a la tabla de conexiones esa entrada y su dual
   //NewConectionTable(pktTCP->ulIPSource,pktTCP->usPortSourceTCP,pktTCP->ulIPDestination,pktTCP-
>usPortDestinationTCP);
   NewConectionTable(pktTCP->ulIPDestination,pktTCP->usPortDestinationTCP,pktTCP->ulIPSource,pktTCP-
>usPortSourceTCP);
   //yo no añadiría la dual,

   newInfo->totalPktCap = 0;
   newInfo->totalPktAda = 0;
   newInfo->buffer = NULL;
   newInfo->reintentosScSc  = 0;



   if (gHashCapturador == NULL){
      newHash          = (hashTable*)calloc(1,sizeof(hashTable));
      newHash->ipSource  = pktTCP->ulIPSource;
      newHash->ptoSource = pktTCP->usPortSourceTCP;
      newHash->info      = newInfo;
      newHash->next      = NULL;
      gHashCapturador    = newHash;
   }
   else {
      //buscar si coincide con alguna entrada en la hash
      encontrado = 0;
      for (auxHash = gHashCapturador; auxHash != NULL ; auxHash = auxHash->next){
         /* Existe una entrada en la hash para esa conexión Origen*/
         if ((auxHash->ipSource == pktTCP->ulIPSource)  && (auxHash->ptoSource == pktTCP->usPortSourceTCP)){
            //Hay una entrada origen, insertar new info como destino
            newInfo->next = auxHash->info;
            auxHash->info = newInfo;
            encontrado = 1;
            break;
         }
      }
      if (encontrado == 0) {
         //No coincide con ninguna entrada de la Hash, creamos una nueva
         newHash          = (hashTable*)calloc(1,sizeof(hashTable));
         newHash->ipSource  = pktTCP->ulIPSource;
         newHash->ptoSource = pktTCP->usPortSourceTCP;
         newHash->info      = newInfo;
         newHash->next      = gHashCapturador;
         gHashCapturador    = newHash;
      }
   }
}


/*****************************************************************************************************
NOMBRE:      NewConectionAdaptador
SINOPSIS:    crea una nueva conexion en la hash del Adaptador
PARAMETROS:  HeaderPacket * (entrada): pkt con la informacion de la conexion
DESCRIPCION: Crea una nueva conexión en la hash del Adaptador con la información
             del pkt recibido y se enlaza con su conexion Dual de la hash del
             Capturador
*****************************************************************************************************/
void NewConectionAdaptador(HeaderPacket *pktTCP, infoHash *infoHashCapt){
   hashTable *newHash, *auxHash;
```

```
   infoHash  *newInfo;
   int encontrado;

   //Almacenamos la información de la conexión
   newInfo                = (infoHash*)calloc(1,sizeof(infoHash));
   newInfo->ipDest        = pktTCP->ulIPSource;
   newInfo->ptoDest       = pktTCP->usPortSourceTCP;
   newInfo->endConection  = (unsigned char)0;
   newInfo->rstRecibido   = 0;
   newInfo->numAsentimiento = pktTCP->ulAckNumber; // El discriminador elige uno aleatorio
   newInfo->numSecPkt     = pktTCP->ulSecuenceNumber + 1 ; //nº secuencia paquete esperado
   newInfo->next          = NULL;
   newInfo->hiloRedirector  = NULL;
   newInfo->hiloCapturador  = NULL;

   newInfo->buffer        = NULL;
   newInfo->totalPktAda   = 0;
   newInfo->totalPktCap   = 0;
   newInfo->reintentosScSc = 0;
   newInfo->tamWin        = pktTCP->usWin;
   newInfo->vCanal        = TIMER_LENTO;

   NewConectionTable(pktTCP->ulIPSource,pktTCP->usPortSourceTCP,pktTCP->ulIPDestination,pktTCP-
>usPortDestinationTCP);
   //Añadimos la nueva conexion a la hash
   if (gHashAdaptador == NULL){
      newHash          = (hashTable*)calloc(1,sizeof(hashTable));
      newHash->ipSource  = pktTCP->ulIPDestination;
      newHash->ptoSource = pktTCP->usPortDestinationTCP;
      newHash->info      = newInfo;
      newHash->next      = NULL;
      gHashAdaptador     = newHash;
   }
   else {
      //buscar si coincide con alguna entrada en la hash
      encontrado = 0;
      for (auxHash = gHashAdaptador; auxHash != NULL ; auxHash = auxHash->next){
         /* Existe una entrada en la hash para esa conexión Origen*/
         if ((auxHash->ipSource == pktTCP->ulIPDestination)  && (auxHash->ptoSource == pktTCP-
>usPortDestinationTCP)){
            //Hay una entrada origen, insertar new info como destino
            newInfo->next = auxHash->info;
            auxHash->info = newInfo;
            encontrado = 1;
            break;
         }
      }
      if (encontrado == 0) {
         //No coincide con ninguna entrada de la Hash, creamos una nueva
         newHash          = (hashTable*)calloc(1,sizeof(hashTable));
         newHash->ipSource  = pktTCP->ulIPDestination;
         newHash->ptoSource = pktTCP->usPortDestinationTCP;
         newHash->info      = newInfo;
         newHash->next      = gHashAdaptador;
         gHashAdaptador     = newHash;
      }
   }

   newInfo->conexDual = infoHashCapt;
   infoHashCapt->conexDual = newInfo;
}


/*****************************************************************************************
NOMBRE:      SendBuffer2Capturador
SINOPSIS:    Envia los pkt almacenados al capturador
PARAMETROS:  infoHash * (entrada): Elemento con la informacion de la conexion y
                el buffer a enviar
          int (salida): Devuelve 0 si hemos enviado un FIN, 1 si no
DESCRIPCION: Envia los pkt almacenados en el buffer al Capturador. Para intentar
          evitar "saturar" a la aplicacion receptora, se envian a rafagas,
          una primera rafaga con el maximo de pkt permitidos por la ventana
          de congestion de recepcion, y el resto con la mitad de la ventana,
          esperando entre rafaga y rafaga 10 milisegundos por paquete.
          Los paquetes se envian mediante una llamada a SendPkt.
```

```
*****************************************************************************************************/
int SendBuffer2Capturador(infoHash *infoHashAdap){
   bufferPkt *aux;
   infoHash  *infoHashCapt;
   unsigned long tWinPkt, i, tWinPktMitad;
   unsigned short inicio;

   infoHashCapt = infoHashAdap->conexDual;
   tWinPkt      = infoHashCapt->tamWin/MSS; //Numero de pkt con tamanyo de datos maximo que caben en la ventana
del receptor
   tWinPktMitad = tWinPkt/2;
   // Si el primer paquete es un SYN, mandamos los paquetes de datos siguientes (no SYN)
   if (infoHashAdap->buffer->pkt.ucFsinc == (unsigned char) 1){
       aux = infoHashAdap->buffer->next;
   }
   // El primer paquete no es un SYN, mandamos todo el buffer entero
   else{
      aux = infoHashAdap->buffer;
   }
   inicio = 0;
   while(aux != NULL){
      //Tengo que suplantar el número de ack
      for (i = inicio; i < tWinPkt; i ++) {
            aux->pkt.ulAckNumber = infoHashAdap->numAsentimiento;
            aux->pkt.usWin = 4*MSS;
            if (aux->pkt.ucFdataEnd == (unsigned char)1 ){
               SendPkt (fileNameEnv,&(aux->pkt));
               MyStat->pktSndCapt++;
               MyStat->bytSndCapt = MyStat->bytSndCapt + aux->pkt.uiLengthData - (aux->pkt.ucDataOffset * 4);
               MyStat->pktSndCaptTCP++;
               MyStat->bytSndCaptTCP = MyStat->bytSndCaptTCP + aux->pkt.uiLengthData - (aux->pkt.ucDataOffset *
4);

               infoHashCapt->numAsentimiento = infoHashCapt->numAsentimiento + (aux->pkt.uiLengthData -aux-
>pkt.ucDataOffset*4) + 1;
               infoHashAdap->endConection = (unsigned char) 1;
               infoHashAdap->hiloCapturador = NULL;
               return 0;
            } else{
               SendPkt (fileNameEnv,&(aux->pkt));
               MyStat->pktSndCapt++;
               MyStat->bytSndCapt = MyStat->bytSndCapt + aux->pkt.uiLengthData - (aux->pkt.ucDataOffset * 4);
               MyStat->pktSndCaptTCP++;
               MyStat->bytSndCaptTCP = MyStat->bytSndCaptTCP + aux->pkt.uiLengthData - (aux->pkt.ucDataOffset *
4);

               infoHashCapt->numAsentimiento = infoHashCapt->numAsentimiento + (aux->pkt.uiLengthData -aux-
>pkt.ucDataOffset*4);
            }
            aux = aux->next;
            if (aux == NULL) {
               infoHashAdap->hiloCapturador = NULL;
               return 1;
            }
      }
      //Ya hemos mandado el tamaño de ventana, esperamos un tiempo
      Sleep((tWinPkt - inicio)*10);//Esperamos 10 milisegundos por paquete enviado
      inicio = tWinPktMitad;
   }
   infoHashAdap->hiloCapturador = NULL;
   return 1;
}

/*****************************************************************************************************
NOMBRE:      SendBuffer2CapturadorDual
SINOPSIS:    Envia los pkt almacenados al capturador
PARAMETROS:  infoHash * (entrada) Elemento con la informacion de la conexion y
                el buffer a enviar
          int (salida): Devuelve 0 si el ultimo pkt es un FIN, 1 si no.
DESCRIPCION:
Igual que la anterior pero para conexiones iniciadas por el lado del capturador
*****************************************************************************************************/
int SendBuffer2CapturadorDual(infoHash *infoHashAdap){
   bufferPkt *aux;
   infoHash  *infoHashCapt;
```

```
   unsigned long tWinPkt, i, tWinPktMitad;
   unsigned short inicio = 0;

   printf("(SendBuffer2CapturadorDual)Inicio\n");
   if (infoHashAdap == NULL){
      printf("(SendBuffer2CapturadorDual)La hash del adaptador pasada es nula\n");
      return -1;
   }
   infoHashCapt = infoHashAdap->conexDual;
   aux          = infoHashAdap->buffer;
   if (infoHashCapt == NULL) {
      printf("(SendBuffer2CapturadorDual) No tenemos informacion de la hash dual\n");
      tWinPkt = infoHashAdap->tamWin/MSS;
   }
   else {
      tWinPkt      = infoHashCapt->tamWin/MSS; //Numero de pkt con tamanyo de datos maximo que caben en la
ventana del receptor
   }
   tWinPktMitad = tWinPkt/2;
   printf("Tamaño de ventana en la hash del capturador: %u\n", infoHashCapt->tamWin);
   printf("Tamaño de ventana en paquetes: %u, tamaño mitad: %u\n", tWinPkt,tWinPktMitad);

   while(aux != NULL){
      printf("(SendBuffer2CapturadorDual)Buffer no nulo\n");
      for (i = inicio; i < tWinPkt; i ++) {
         printf("(SendBuffer2CapturadorDual)Iteracion numero: %u\n",i);
         aux->pkt.ulAckNumber      = infoHashCapt->numSecPkt;
         aux->pkt.ulSecuenceNumber = infoHashCapt->numAsentimiento;
         aux->pkt.usWin = 4*MSS;
         if (aux->pkt.ucFdataEnd == (unsigned char)1 ){
            //printf("Enviando un FIN\n");
            printf("(SendBuffer2CapturadorDual)Enviando a SendPkt FIN\n");
            SendPkt (fileNameEnv,&(aux->pkt));
            MyStat->pktSndCapt++;
            MyStat->bytSndCapt = MyStat->bytSndCapt + aux->pkt.uiLengthData - (aux->pkt.ucDataOffset * 4);
            MyStat->pktSndCaptTCP++;
            MyStat->bytSndCaptTCP = MyStat->bytSndCaptTCP + aux->pkt.uiLengthData - (aux->pkt.ucDataOffset * 4);
            infoHashCapt->numAsentimiento = infoHashCapt->numAsentimiento + (aux->pkt.uiLengthData -aux-
>pkt.ucDataOffset*4) + 1;
            //El adaptador envia un Fin al capturador
            infoHashAdap->endConection = (unsigned char) 1;
            infoHashAdap->hiloCapturador = NULL;
            return 0;
         } else {
            printf("(SendBuffer2CapturadorDual)Enviando a SendPkt\n");
            SendPkt (fileNameEnv,&(aux->pkt));
            MyStat->pktSndCapt++;
            MyStat->bytSndCapt = MyStat->bytSndCapt + aux->pkt.uiLengthData - (aux->pkt.ucDataOffset * 4);
            MyStat->pktSndCaptTCP++;
            MyStat->bytSndCaptTCP = MyStat->bytSndCaptTCP + aux->pkt.uiLengthData - (aux->pkt.ucDataOffset * 4);
            infoHashCapt->numAsentimiento = infoHashCapt->numAsentimiento + (aux->pkt.uiLengthData -aux-
>pkt.ucDataOffset*4);
         }
         aux = aux->next;
         if (aux == NULL) {
            infoHashAdap->hiloCapturador = NULL;
            return 1;
         }
      }
      Sleep((tWinPkt - inicio)*10);//Esperamos 10 milisegundos por paquete enviado
      inicio = tWinPktMitad;
   }
   infoHashAdap->hiloCapturador = NULL;
   return 1;
}


/*****************************************************************************************************
NOMBRE:      SendBuffer2Redirector
SINOPSIS:    Envia los pkt almacenados al redirector
PARAMETROS:  infoHash * (entrada): elemento con la informacion de la conexion y
                el buffer a enviar
             int (salida): 0 si ha habido un error, 1 si no hay errores
DESCRIPCION: Envia al REDIRECTOR los paquetes almacenados en el buffer. Se envia
             un numero de paquetes indicado en la informacion de la hash
```

```
**********************************************************************************************************/

int SendBuffer2Redirector(infoHash * auxInfo){
   bufferPkt *aux;
   int contador;

   aux = auxInfo->buffer;
   for (contador = 1; contador <= auxInfo->totalPktAda; contador++) {
      if (aux == NULL) {
         printf("Error: queremos enviar mas paquetes de los almacenados\n");
         auxInfo->hiloRedirector = NULL;
         return 0;
      }

      aux->pkt.usNumPacTotal = auxInfo->totalPktAda;
      printf("Enviando al Redirector el pkt %d de %d\n", aux->pkt.usPacActual, aux->pkt.usNumPacTotal);
      SendPkt (fileNameRedirector,&(aux->pkt));
      MyStat->pktSndRed++;
      MyStat->bytSndRed = MyStat->bytSndRed + aux->pkt.uiLengthData - (aux->pkt.ucDataOffset * 4);
      MyStat->pktSndRedTCP++;
      MyStat->bytSndRedTCP = MyStat->bytSndRedTCP + aux->pkt.uiLengthData - (aux->pkt.ucDataOffset * 4);

      aux = aux->next;
   }
   auxInfo->hiloRedirector = NULL;
   return 1;
}


/*****************************************************************************
NOMBRE:      SendPkt
SINOPSIS:    Envia un pkt al destino (Capturador/Redirector) indicado
PARAMETROS:  AnsiString     (entrada): Destino al que enviar
             HeaderPacket * (entrada): Pkt a enviar
DESCRIPCION:


*****************************************************************************/
void SendPkt (AnsiString  fileName,HeaderPacket * pkt){

   //HANDLE fileWrite;
   DWORD  dwWaitResult1;
   //char * Memory;
   struct pseudo p;
   //struct in_addr IpOrigen, IpDestino;
   DWORD dwWaitResult;

   //IpOrigen.S_un.S_addr  = pkt->ulIPSource;
   //IpDestino.S_un.S_addr = pkt->ulIPDestination;

   printf("(SendPkt) inicio\n");
   /*Semáforo para escribir al capturador*/
   if ((hSemNewWrite = CreateSemaphore(NULL,0,1,"SemNewWrite")) == NULL){
      printf("ERROR: No se ha podido abrir SemNewWrite\n");
      strcpy(MyAlarmas.cId ,"Discriminador");
      strcpy(MyAlarmas.cTxtAlarm,"[ERROR] Error en la creación de interfaces");
      dwWaitResult = WaitForSingleObject(infoAlarmas.SemWrite,3000);
      switch(dwWaitResult){

         case WAIT_OBJECT_0:
            memcpy(infoAlarmas.Memo,&MyAlarmas, sizeof(MyAlarmas));
            ReleaseSemaphore(infoAlarmas.SemRead,1,NULL);
            break;

         case WAIT_TIMEOUT:
            printf("Time Out 1 hilo\n");
            return;

         default:
            printf("failed 1 hilo\n");
            break;
      }//fin Switch

      exit(0);
   }
```

```
   /*Creamos el semáforo para recibir permiso de escritura del capturador*/
   if ((hSemFinWrite = CreateSemaphore(NULL,1,1,"SemFinWrite")) == NULL){
      printf("ERROR: NO se ha podido abrir el Semáforo SemFinWrite\n");
      strcpy(MyAlarmas.cId ,"Discriminador");
      strcpy(MyAlarmas.cTxtAlarm,"[ERROR] Error en la creación de interfaces");
      dwWaitResult = WaitForSingleObject(infoAlarmas.SemWrite,3000);
      switch(dwWaitResult){

         case WAIT_OBJECT_0:
            memcpy(infoAlarmas.Memo,&MyAlarmas, sizeof(MyAlarmas));
            ReleaseSemaphore(infoAlarmas.SemRead,1,NULL);
            break;

         case WAIT_TIMEOUT:
            printf("Time Out 1 hilo\n");
            return;

         default:
            printf("failed 1 hilo\n");
            break;
      }//fin Switch

      exit(0);
   }
/*
   if (pkt->usIdentification == Ident){
      return;
   }
*/

   if (fileName == fileNameEnv){
      //Pido permiso para escribir en memoria compartida
      //dwWaitResult1 = WaitForSingleObject(hSemFinWrite, INFINITE);
      dwWaitResult1 = WaitForSingleObject(hSemFinWrite, 1000);
      //Lo ponemos a medio segundo
      OutputDebugStringA("\n----->El capturador me deja escribir en memoria");
      /*Se lo mando al Capturador en memoria compartida*/
      switch(dwWaitResult1){
         case WAIT_OBJECT_0:
            memcpy(MemoryWrite,pkt,sizeof(struct HeaderPacket));
            printf("EScrito paquete en la memoria del Capturador\n");
            /*Guardamos el ID del que acabamos de Mandar*/
            Ident = pkt->usIdentification;
            /*Avisamos al Capturador de que hay una nueva trama para enviar a la red*/
            printf("Paquete escrito en la memoria compartida con el capturador\n");
            if (ReleaseSemaphore(hSemNewWrite,1,NULL) == 0) {
              strcpy(MyAlarmas.cId ,"Discriminador");
              strcpy(MyAlarmas.cTxtAlarm,"[ERROR] Posible error de comunicación interno");
              dwWaitResult = WaitForSingleObject(infoAlarmas.SemWrite,3000);
              switch(dwWaitResult){

                case WAIT_OBJECT_0:
                  memcpy(infoAlarmas.Memo,&MyAlarmas, sizeof(MyAlarmas));
                  ReleaseSemaphore(infoAlarmas.SemRead,1,NULL);
                  break;

                case WAIT_TIMEOUT:
                  printf("Time Out 1 hilo\n");
                  return;

                default:
                  printf("failed 1 hilo\n");
                  break;
               }//fin Switch
            }
            break;

        case WAIT_TIMEOUT:
            printf("TIME OUT Permiso del Capturador para escritura\n");
          /*  MyStat->pktDescartados++;
            if (pkt->ucProtocol == (unsigned char) 6) {
               MyStat->pktDescTCP++;
               MyStat->bytDescTCP     = MyStat->bytDescTCP     + pkt->uiLengthData - (pkt->ucDataOffset*4);
               MyStat->bytDescartados = MyStat->bytDescartados + pkt->uiLengthData - (pkt->ucDataOffset*4);
```

```
            }
            else if (pkt->ucProtocol == (unsigned char) 17) {
               MyStat->pktDescUDP++;
               MyStat->bytDescUDP     = MyStat->bytDescUDP     + pkt->usLengthUDP - 8;
               MyStat->bytDescartados = MyStat->bytDescartados + pkt->usLengthUDP - 8;
            }
            */
            break;
      }//fin switch
   }
   else{

      p.prior = 0;
      p.header = *pkt;

      //Se lo pasamos al redirector por memoria compartida
      dwWaitResult1 = WaitForSingleObject(hSemRedDis,5000);
      switch(dwWaitResult1){
        case WAIT_OBJECT_0:

            memcpy(MemoDisRed,&p,sizeof(p));
            printf("EScrito paquete en la memoria del Redirector\n");
            if (ReleaseSemaphore(hSemDisRed,1,NULL) == 0) {
               strcpy(MyAlarmas.cId ,"Discriminador");
               strcpy(MyAlarmas.cTxtAlarm,"[ERROR] Posible error de comunicación interno");
               dwWaitResult = WaitForSingleObject(infoAlarmas.SemWrite,3000);
               switch(dwWaitResult){

                 case WAIT_OBJECT_0:
                    memcpy(infoAlarmas.Memo,&MyAlarmas, sizeof(MyAlarmas));
                    ReleaseSemaphore(infoAlarmas.SemRead,1,NULL);
                    break;

                 case WAIT_TIMEOUT:
                    printf("Time Out 1 hilo\n");
                    return;

                 default:
                    printf("failed 1 hilo\n");
                    break;
               }//fin Switch
            }
        case WAIT_TIMEOUT:
            //printf("TIME OUT Permiso del Redirector para escritura\n");
            /*
            MyStat->pktDescartados++;
            if (pkt->ucProtocol == (unsigned char) 6) {
               MyStat->pktDescTCP++;
               MyStat->bytDescTCP     = MyStat->bytDescTCP     + pkt->uiLengthData - (pkt->ucDataOffset*4);
               MyStat->bytDescartados = MyStat->bytDescartados + pkt->uiLengthData - (pkt->ucDataOffset*4);
            }
            else if (pkt->ucProtocol == (unsigned char) 17) {
               MyStat->pktDescUDP++;
               MyStat->bytDescUDP     = MyStat->bytDescUDP     + pkt->usLengthUDP - 8;
               MyStat->bytDescartados = MyStat->bytDescartados + pkt->usLengthUDP - 8;
            }
            */
            break;
      }//fin switch
   }
}

/*****************************************************************************
NOMBRE:      DeleteConectionCapturador
SINOPSIS:    Elimina una conexion de la hash del capturador
PARAMETROS:  unsigned long  (entrada): direccion IP origen
             unsigned short (entrada): puerto origen
             unsigned long  (entrada): direccion IP destino
             unsigned short (entrada): puerto destino
             int            (salida) : devuelve 0 si ha encontrado la conexion,
                                       1 si no.
DESCRIPCION: Con la informacion pasada como parametro, busca en la hash del
             Capturador si existe esa conexion. Si la encuentra mata los timer
             y la elimina. Si no la encuentra no hace nada.
*****************************************************************************/
```

```
int DeleteConectionCapturador (unsigned long IPSource,unsigned short usPortSourceTCP,unsigned long
ulIPDestination,unsigned short usPortDestinationTCP){

   hashTable *auxHash, *auxHashAnt, *hash2Delete;
   infoHash  *auxInfo, *auxInfoAnt, *info2Delete, *hashDual;

   //Buscamos IP Origen - Puerto Origen
   auxHashAnt = NULL;

   for (auxHash = gHashCapturador; auxHash != NULL ; auxHashAnt = auxHash, auxHash = auxHash->next){
      if (auxHash->ipSource == IPSource  &&  auxHash->ptoSource == usPortSourceTCP){
         auxInfoAnt = NULL;
         //Buscamos IP Destino - Puerto Destino
         for (auxInfo = auxHash->info; auxInfo != NULL; auxInfoAnt = auxInfo, auxInfo = auxInfo->next){
            if (auxInfo->ipDest == ulIPDestination && auxInfo->ptoDest == usPortDestinationTCP){
               DeleteBuffer(auxInfo,0);
               timeKillEvent(auxInfo->timerScSc);
               timeKillEvent(auxInfo->timerID);
               if (auxInfo->hiloCapturador != NULL) {
                  TerminateThread(auxInfo->hiloCapturador,0);
                  auxInfo->hiloCapturador = NULL;
               }
               if(auxInfo->hiloRedirector != NULL) {
                  TerminateThread(auxInfo->hiloRedirector,0);
                  auxInfo->hiloRedirector = NULL;
               }
               info2Delete = auxInfo;

               //Primer elemento de informacion
               if (auxInfoAnt == NULL){
                  auxHash->info = auxInfo->next;
               }
               //Cualquier elemento salvo el primero
               else {
                  auxInfoAnt->next = auxInfo->next;
               }
               hashDual = info2Delete->conexDual;
               if (hashDual != NULL) {
                  hashDual->conexDual = NULL;
                  MyStat->numConexTCPactivas--;
               }
               free(info2Delete);

               // Borramos las conexiones de la tabla de conexiones
               DeleteConnectionTable(IPSource,usPortSourceTCP,ulIPDestination,usPortDestinationTCP);
               DeleteConnectionTable(ulIPDestination,usPortDestinationTCP,IPSource,usPortSourceTCP);

               // Actualizamos IP Origen-Pto Origen
               if (auxHash->info == NULL){
                  hash2Delete = auxHash;
                  //Primer elemento de la hash
                  if (auxHashAnt == NULL){
                     gHashCapturador = gHashCapturador->next;
                  }
                  else{
                     auxHashAnt->next = auxHash->next;
                  }
                  hash2Delete->next = NULL;
                  free (hash2Delete);
               }
               return 0;
            }
         }
      }
   }
   return 1;
}

/*****************************************************************************
NOMBRE:      DeleteConectionadaptador
SINOPSIS:    Elimina una conexion de la hash del Adaptador
PARAMETROS:  unsigned long  (entrada): direccion IP origen
             unsigned short (entrada): puerto origen
             unsigned long  (entrada): direccion IP destino
             unsigned short (entrada): puerto destino
```

```
        int         (salida) : devuelve 0 si ha encontrado la conexion,
                               1 si no.
DESCRIPCION: Con la informacion pasada como parametro, busca en la hash del
        Adaptador si existe esa conexion. Si la encuentra mata los timer
        y la elimina. Si no la encuentra no hace nada.
*****************************************************************************/
int DeleteConectionAdaptador (unsigned long IPSource,unsigned short usPortSourceTCP,unsigned long
ulIPDestination,unsigned short usPortDestinationTCP){

   hashTable *auxHash, *auxHashAnt, *hash2Delete;
   infoHash  *auxInfo, *auxInfoAnt, *info2Delete, *hashDual;

   //Buscamos IP Origen - Puerto Origen
   auxHashAnt = NULL;
   for (auxHash = gHashAdaptador; auxHash != NULL; auxHashAnt = auxHash, auxHash = auxHash->next){
      if (auxHash->ipSource == IPSource && auxHash->ptoSource == usPortSourceTCP){
         auxInfoAnt = NULL;
         //Buscamos IP Destino - Puerto Destino
         for (auxInfo = auxHash->info; auxInfo != NULL; auxInfoAnt = auxInfo, auxInfo = auxInfo->next){
            if (auxInfo->ipDest == ulIPDestination && auxInfo->ptoDest == usPortDestinationTCP){
               timeKillEvent(auxInfo->timerScSc);
               timeKillEvent(auxInfo->timerID);
               if (auxInfo->hiloCapturador != NULL) {
                  TerminateThread(auxInfo->hiloCapturador,0);
                  auxInfo->hiloCapturador = NULL;
               }
               if (auxInfo->hiloRedirector != NULL) {
                  TerminateThread(auxInfo->hiloRedirector,0);
                  auxInfo->hiloRedirector = NULL;
               }
               DeleteBuffer(auxInfo,0);
               info2Delete = auxInfo;
               //Primer elemento de informacion
               if (auxInfoAnt == NULL){
                  auxHash->info = auxInfo->next;
               }
               //Cualquier elemento salvo el primero
               else{
                  auxInfoAnt->next = auxInfo->next;
               }
               hashDual = info2Delete->conexDual;
               if (hashDual != NULL){
                   hashDual->conexDual = NULL;
                   MyStat->numConexTCPactivas--;
               }
               free(info2Delete);

               // Borramos las conexiones de la tabla de conexiones
               DeleteConnectionTable(IPSource,usPortSourceTCP,ulIPDestination,usPortDestinationTCP);
               DeleteConnectionTable(ulIPDestination,usPortDestinationTCP,IPSource,usPortSourceTCP);

               // Actualizamos IP Origen-Pto Origen en el caso de que esa entrada ya no tenga conexiones destino
               if (auxHash->info == NULL){
                  hash2Delete = auxHash;
                  // Primer elemento de la hash: si es el 1er elemento,
                  // cambiamos el puntero a la hash global
                  if (auxHashAnt == NULL){
                     gHashAdaptador = gHashAdaptador->next;
                  }
                  else{
                     auxHashAnt->next = auxHash->next;
                  }
                  hash2Delete->next = NULL;
                  free(hash2Delete);
               }
               return 0;
            }
         }
      }
   }
   return 1;
}

/*****************************************************************************
NOMBRE:     DeleteHashCapturador
```

```
SINOPSIS:    Elimina la hash del capturador
PARAMETROS:
DESCRIPCION: Elimina la hash del Capturador, todos sus elementos. Por cada
          conexion envia un RST al Capturador
*****************************************************************************/
void DeleteHashCapturador () {

   hashTable *auxHash;
   infoHash  *auxInfo;

   for(auxHash = gHashCapturador; auxHash != NULL ; auxHash=auxHash->next){
      for(auxInfo = auxHash->info; auxInfo != NULL; auxInfo = auxInfo->next){
         DeleteConnectionTable(auxHash->ipSource,auxHash->ptoSource,auxInfo->ipDest,auxInfo->ptoDest);
         //mandar un Reset al Capturador
         CreateSendResetHash(auxHash, auxInfo,1);
         DeleteConectionCapturador(auxHash->ipSource,auxHash->ptoSource,auxInfo->ipDest,auxInfo->ptoDest);
      }
   }
}

/*****************************************************************************
NOMBRE:      DeleteHashAdaptador
SINOPSIS:    Elimina la hash del adaptador
PARAMETROS:
DESCRIPCION: Elimina la hash del Adaptador, todos sus elementos. Por cada
          conexion envia un RST al Redirector (ServCom destino)
*****************************************************************************/
void DeleteHashAdaptador () {

   hashTable *auxHash;
   infoHash  *auxInfo;

   for(auxHash = gHashAdaptador; auxHash != NULL ; auxHash=auxHash->next){
      for(auxInfo = auxHash->info; auxInfo != NULL; auxInfo = auxInfo->next){
         DeleteConnectionTable(auxHash->ipSource,auxHash->ptoSource,auxInfo->ipDest,auxInfo->ptoDest);
         //mandar un RST al Redirector
         CreateSendResetHash(auxHash, auxInfo,0);
         DeleteConectionAdaptador(auxHash->ipSource,auxHash->ptoSource,auxInfo->ipDest,auxInfo->ptoDest);
      }
   }

}



/*****************************************************************************
NOMBRE:      DeleteBuffer
SINOPSIS:    Elimina el buffer de pkt de una conexion
PARAMETROS:  infoHash * (entrada): Elemento con la informacion de la conexion y
                el buffer a borrar
          bool (entrada): Indica si hay que comprobar el cierre de conexiones
          int (salida): Devuelve -1 si ha habido errores, 1 si el ultimo pkt
                es un FIN, 0 si no lo es
DESCRIPCION: Se usa para borrar un buffer de pkt, se llama a esta funcion
          cuando se ha recibido un ACK-Sc-Sc confirmando que el otro extremo
          ha recibido todos los pkt o cuando borramos una conexion.
          Recorre todo el buffer y va eliminando elementos. Si la opcion de
          fin esta activa, se comprueba si el ultimo pkt es un FIN, en cuyo
          caso se manda un ACK al Capturador del FIN y se comprueba si el
          otro extremo ha cerrado ya la conexion, si es asi se lanza un timer
          para la liberacion de la conexion
*****************************************************************************/

//siempre se usa en la hash del Capturador (y cuando liberamos conexiones en ambas)
int DeleteBuffer (infoHash *info, bool fin) {
   bufferPkt *auxBuffer;
   infoHash  *infoDual;
   openConnection *auxFin;

   auxBuffer = info->buffer;
   //Borramos todos los paquetes menos el ultimo
   if (auxBuffer == NULL) {
      return -1;
   }
   while(auxBuffer->next != NULL){
```

```
        info->buffer = auxBuffer->next;
        free(auxBuffer);
        auxBuffer = info->buffer;
    }

    if (fin) {
        //Si fin esta activo, comprobamos si el ultimo paquete  es un fin
        // Para tratar el cierre de conexiones
        if (auxBuffer->pkt.ucFdataEnd == (unsigned char) 1){
            //Si el ultimo pkt que borro es un fin, tengo que enviar
            // a la aplicacion el ack del fin
            CreateAckFin(&(auxBuffer->pkt),info->numAsentimiento);
            info->numSecPkt++;
            //Ya le he confirmado el fin a mi Aplicacion
            info->endConection = (unsigned char) 2;
            infoDual = info->conexDual;
            if (infoDual == NULL) {
                printf("Error.... (DeleteBuffer) no tengo acceso a la hash dual\n");
                return -1;
            }
            if (infoDual->endConection == (unsigned char) 2) {
                //Lanzar un Timer para borrar esto
                auxFin = (openConnection *) calloc(1,sizeof(openConnection));
                auxFin->ipSource  = auxBuffer->pkt.ulIPSource;
                auxFin->ipDest    = auxBuffer->pkt.ulIPDestination;
                auxFin->ptoSource = auxBuffer->pkt.usPortSourceTCP;
                auxFin->ptoDest   = auxBuffer->pkt.usPortDestinationTCP;
                info->timerID     = timeSetEvent(TIME_OVER_SC, 0,IntFIN, (unsigned long)auxFin, TIME_ONESHOT);
                return 1;
            }
        }
    }
    free(auxBuffer);
    info->buffer = NULL;
    return 0;
}

/*****************************************************************************
NOMBRE:      DeletePktBuffer
SINOPSIS: Borra pkt de un buffer que ya han sido asentidos
PARAMETROS:  infoHash * (entrada): Elemento con la informacion de la conexion y
                        el buffer a borrar
           unsigned long (entrada): Numero de asentimiento recibido
DESCRIPCION: Recibido un ACK, recorremos el buffer para borrar los paquetes que
           ya han sido asentidos, aquellos cuyo numero de secuencia sea menor
           que el numero de ACK recibido. Si el ACK recibido es igual al
           numero de secuencia del primer paquete del buffer, lo rtx.
*****************************************************************************/

// Siempre la llamamos para borrar de la Hash del Adaptador
// cuando hemos recibido un Ack del Capturador
void DeletePktBuffer (infoHash *auxInfo, unsigned long numACK){
    bufferPkt *aux;
    infoHash  *infoDual;
    openConnection  *auxFin;

    aux = auxInfo->buffer;
    if (aux == NULL) {
        printf("(DeletePktBuffer) buffer nulo, nada que borrar\n");
        return;
    }
    //Si el ack recibido es igual al numero de seq del paquete, es que nos esta pidiendo una Rtx
    if (auxInfo->buffer->pkt.ulAckNumber == numACK){
        printf("(DeletePktBuffer) Rtx\n");
        SendPkt(fileNameEnv,&(auxInfo->conexDual->buffer->pkt));
        return;
    }
    while (aux != NULL){
        if (aux->pkt.ulSecuenceNumber < numACK){
            if (aux->pkt.ucFdataEnd == (unsigned char) 1) {
                auxInfo->totalPktCap = 0;
                //Enviar Ack Sc-Sc
                //Como es Ack del fin, es importante que llegue-> lo envio 3 veces
                printf("(DeletePktBuffer) enviando ack-fin SC-SC\n");
                CreateSendAck_SC_SC(&(aux->pkt), aux->pkt.usNumPacTotal + 1);
```

```
            CreateSendAck_SC_SC(&(aux->pkt), aux->pkt.usNumPacTotal + 1);
            CreateSendAck_SC_SC(&(aux->pkt), aux->pkt.usNumPacTotal + 1);
            //El Capturador me asiente el FIN
            auxInfo->endConection = (unsigned char) 2;
            auxInfo->numSecPkt++;
            infoDual = auxInfo->conexDual;
            if (infoDual == NULL) {
               printf("Error..... No tenemos conexion dual\n");
               free(aux);
               auxInfo->buffer = NULL;
               return;
            }
            if (infoDual->endConection == (unsigned char) 2) {
               //Borrar conexiones
               //Lanzar un timer para borrar esto
               printf("Conexion cerrada en los dos extremos, lanzando timer FIN\n");
               auxFin = (openConnection *) calloc(1,sizeof(openConnection));
               auxFin->ipSource  = aux->pkt.ulIPDestination;
               auxFin->ipDest    = aux->pkt.ulIPSource;
               auxFin->ptoSource = aux->pkt.usPortDestinationTCP;
               auxFin->ptoDest   = aux->pkt.usPortSourceTCP;
               infoDual->timerID = timeSetEvent(TIME_OVER_SC, 0,IntFIN, (unsigned long)auxFin, TIME_ONESHOT);
               return;
            }
            //Damos por hecho que el FIN siempre es el ultimo pkt del buffer
            free(aux);
            auxInfo->buffer = NULL;
            return;
         }
         if (aux->prev == NULL && aux->next == NULL){
            //Aqui ya hemos recibido confirmacion de todos los paquetes que teniamos
            //en el buffer del adaptador, por lo que podemos enviarle el ACK_SC_SC
            //Enviar Ack Sc-Sc
            auxInfo->totalPktCap = 0;
            printf("(DeletePktBuffer) confirmados todos los paquetes, enviando ACK SC-SC\n");
            CreateSendAck_SC_SC(&(aux->pkt), aux->pkt.usNumPacTotal + 1);
            free(aux);
            auxInfo->buffer = NULL;
            return;
         }
         auxInfo->buffer = aux->next;
         aux->next = NULL;
         free(aux);
         auxInfo->buffer->prev = NULL;
         aux = auxInfo->buffer;
      }
      else{
         printf("El siguiente pkt no ha sido confirmado\n");
         return;
      }
   }
}

/*****************************************************************************
NOMBRE:      MergeBufferCapturador
SINOPSIS:    Recibe un paquete y un elemento infoHash y ordena el pkt en buffer
             de infoHash
PARAMETROS:  HeaderPacket * (entrada): pkt a insertar
             infoHash * (entrada): Elemento con la informacion de la conexion y
                          el buffer de pkt
             BufferPkt * (salida): puntero al buffer ordenado
DESCRIPCION: Comprueba el estado del buffer y almacena el pkt si este es correcto:
             Si su numero de secuencia coincide con el esperado-> por tanto siempre
             inserta al final del buffer.
             Devuelve un puntero al inicio del buffer ordenado.
*****************************************************************************/

bufferPkt* MergeBufferCapturador (infoHash *info, HeaderPacket *pkt){

   bufferPkt *newPktBuff, *buff, *aux;
   infoHash  *infoHashDual;

   buff        = info->buffer;
   infoHashDual = info->conexDual;
```

```
// Creamos el nodo nuevo en el que se encontrará el  valor a añadir */
newPktBuff      = (bufferPkt*)calloc(1,sizeof(bufferPkt));
newPktBuff->pkt = *pkt;
//No existe el buffer, el paquete a añadir es el primero
if (buff == NULL) {
   //Solo deberia estar vacio si ya ha vencido el TIMER y vuelve a enviar datos
   // o si el otro extremo es el que ha iniciado la conexion
   newPktBuff->prev = NULL;
   newPktBuff->next = NULL;
   if (pkt->ucFdataEnd == 1) {
      info->endConection = (unsigned char) 1;
   }
   info->totalPktCap++;
   newPktBuff->pkt.usPacActual = info->totalPktCap;//Posicion relativa del paquete
   info->numSecPkt         = info->numSecPkt + (pkt->uiLengthData - pkt->ucDataOffset*4);
   infoHashDual->numSecPkt = infoHashDual->numSecPkt + (pkt->uiLengthData - pkt->ucDataOffset*4);
   if (infoHashDual->buffer != NULL) {
      CheckAckAdaptador(pkt);
   }
   CreateSendAck(pkt, info, true, fileNameEnv);
   return newPktBuff ;
}
// Existe el buffer, debemos meter el paquete de forma ordenada
aux = buff;
//Insertamos el paquete de forma ordenada creciente
while (aux != NULL){
   // Recibimos un paquete duplicado, lo descartamos
   if (pkt->ulSecuenceNumber == aux->pkt.ulSecuenceNumber){
      free(newPktBuff);
      return buff;
   }
   // Nº secuencia paquete recibido menor que el del buffer
   // Lo insertamos antes que ese elmento
   if (pkt->ulSecuenceNumber < aux->pkt.ulSecuenceNumber){
      printf("\n\n++++++++++++++++\nAqui no debemos entrar\n");
      //estamos en el primer nodo
      if (aux->prev == NULL){
         //El paquete recibido es el esperado
         if (info->numSecPkt == pkt->ulSecuenceNumber){
            newPktBuff->next = aux;
            newPktBuff->prev = NULL;
            aux->prev = newPktBuff;

            if (pkt->ucFdataEnd == 1) {
               info->endConection = (unsigned char)1;
            }
            info->totalPktCap++;
            newPktBuff->pkt.usPacActual = info->totalPktCap;//Posicion relativa del paquete
            info->numSecPkt         = info->numSecPkt + (pkt->uiLengthData - pkt->ucDataOffset*4);
            infoHashDual->numSecPkt = infoHashDual->numSecPkt + (pkt->uiLengthData - pkt->ucDataOffset*4);
            //Comprobamos si hay más paquetes en el buffer
         }
         /*MANDAR ACK (INFO->NUMSECPKT)*/
         CreateSendAck(pkt, info, true, fileNameEnv);
         return newPktBuff;
      }
      //Insertamos el paquete delante de un nodo intermedio
      else{
         if (info->numSecPkt == pkt->ulSecuenceNumber ){
            newPktBuff->prev  = aux->prev;
            newPktBuff->next  = aux;
            (aux->prev)->next = newPktBuff; /* ADD */
            aux->prev         = newPktBuff;

           if (pkt->ucFdataEnd == 1) {
              info->endConection = (unsigned char) 1;
           }

            info->totalPktCap++;
            newPktBuff->pkt.usPacActual = info->totalPktCap;//Posicion relativa del paquete
            info->numSecPkt             = info->numSecPkt + (pkt->uiLengthData - pkt->ucDataOffset*4);
            infoHashDual->numSecPkt     = infoHashDual->numSecPkt + (pkt->uiLengthData - pkt->ucDataOffset*4);
         }
         /*MANDAR ACK (INFO->NUMSECPKT)*/
         CreateSendAck(pkt, info, true, fileNameEnv);
```

```
            return buff;
         }
      }
      //El paquete recibido es mayor
      else{
         //printf("\n\n+++++++++++++++\nAqui SI debemos entrar\n");
         if (aux->next == NULL){ //el nodo es el último
            if (info->numSecPkt == pkt->ulSecuenceNumber ){
               /* El paquete recibido es mayor que todos, se añade al final*/
               newPktBuff->prev = aux;
               aux->next = newPktBuff;
               newPktBuff->next = NULL; /* ADD */

               if (pkt->ucFdataEnd == 1) {
                  info->endConection = (unsigned char) 1;
               }
               info->numSecPkt            = info->numSecPkt + (pkt->uiLengthData - pkt->ucDataOffset*4);
               info->totalPktCap++;
               newPktBuff->pkt.usPacActual = info->totalPktCap;//Posicion relativa del paquete
               infoHashDual->numSecPkt     = infoHashDual->numSecPkt + (pkt->uiLengthData - pkt->ucDataOffset*4);
               if (infoHashDual->buffer != NULL) {
                  CheckAckAdaptador(pkt);
               }

            }
            else {
               CreateSendAck(pkt,info,true,fileNameEnv);
               CreateSendAck(pkt,info,true,fileNameEnv);
               free(newPktBuff);
            }
            CreateSendAck(pkt, info, true, fileNameEnv);
            return buff;
         }
         else{
            aux = aux->next;
         }
      }
   }//while
   return NULL;
} //fin funcion


/*****************************************************************************
NOMBRE:      MergeBufferAdaptador
SINOPSIS:    Recibe un paquete y un elemento infoHash y ordena el pkt en buffer
             de infoHash
PARAMETROS:  HeaderPacket * (entrada): pkt a insertar
             infoHash * (entrada): Elemento con la informacion de la conexion y
                          el buffer de pkt
             BufferPkt * (salida): puntero al buffer ordenado
DESCRIPCION: Almacena el pkt en el buffer siguiendo el orden indicado en el campo
             usPacActual, su posicion relativa dentro de la rafaga de paquetes que
             tenemos que recibir. Si el pkt no es un duplicado, se almacena en el
             buffer y se decrementa el num de pkt a esperar.
             Devuelve un puntero al inicio del buffer.
*****************************************************************************/
bufferPkt* MergeBufferAdaptador (infoHash *info, HeaderPacket *pkt){

   bufferPkt *newPktBuff, *buff, *aux;
   infoHash *auxInfo;

   buff = info->buffer;

   printf("(MergeBufferAdaptador) inicio\n");
   // Creamos el nodo nuevo en el que se encontrará el  valor a añadir
   newPktBuff = (bufferPkt*)calloc(1,sizeof(bufferPkt));
   newPktBuff->pkt = *pkt;
   info->totalPktAda--;

   //No existe el buffer, el paquete a añadir es el primero
   if (buff == NULL) {
      printf("(MergeBufferAdaptador) buffer nulo\n");
      //Comprobar que no es una Rtx
      auxInfo = info->conexDual;
      if ((pkt->ulSecuenceNumber + (pkt->uiLengthData - pkt->ucDataOffset*4)) == auxInfo->numAsentimiento) {
```

```
            printf("(MergeBufferAdaptador) paquete ya procesado\n");
        //Con este if comprobamos que no sea el ultimo pkt, que es lo que envia el otro
        //Sc cuando no le llega el ACK de todos los pkt
            if (pkt->ucFdataEnd != 1) {
               CreateSendAck_SC_SC(pkt, pkt->usNumPacTotal + 1);
               free(newPktBuff);
               info->totalPktAda = 0;
               return NULL;
            }
        }

        newPktBuff->prev = NULL;
        newPktBuff->next = NULL;
        printf("(MergeBufferAdaptador) nuevo paquete en el buffer\n");
        return newPktBuff ;
    }
    // Existe el buffer, debemos meter el paquete de forma ordenada
    aux = buff;
    //Insertamos el paquete de forma ordenada creciente
    while(aux != NULL){
        // Recibimos un paquete duplicado, lo descartamos
        if (pkt->usPacActual == aux->pkt.usPacActual){
            printf("(MergeBufferAdaptador) paquete duplicado\n");
            info->totalPktAda++;
            free(newPktBuff);
            return buff;
        }

        // Nº secuencia paquete recibido menor que el del buffer
        // Lo insertamos antes que ese elmento
        if (pkt->usPacActual < aux->pkt.usPacActual){
            //estamos en el primer nodo
            if (aux->prev == NULL){
                newPktBuff->next = aux;
                newPktBuff->prev = NULL;
                aux->prev        = newPktBuff;
                info->buffer     = newPktBuff;
                printf("(MergeBufferAdaptador) nuevo paquete en el Inicio del buffer\n");
                return newPktBuff;
            }
            else{
                newPktBuff->prev = aux->prev;
                newPktBuff->next = aux;
                (aux->prev)->next = newPktBuff;
                aux->prev = newPktBuff;
                printf("(MergeBufferAdaptador) nuevo paquete en el buffer\n");
                return buff;
            }
        }
        else{
            // El paquete recibido es mayor que todos, se añade al final*/
            if (aux->next == NULL){
                newPktBuff->prev = aux;
                aux->next = newPktBuff;
                newPktBuff->next = NULL;
                printf("(MergeBufferAdaptador) nuevo paquete en el final del buffer\n");
                return buff;
            }
            //Seguimos buscando en el buffer un paquete menor
            else{
                aux = aux->next;
            }
        }
    }//while
    printf("(MergeBufferAdaptador) paquete sin insertar\n");
    return NULL;
} //fin funcion


/*****************************************************************************
NOMBRE:      getNumSecuence
SINOPSIS:    Obtiene el Numero de Secuencia del otro extremo de la conexion
PARAMETROS:  HeaderPacket * (entrada): pkt a analizar
             infoHash * (salida): Elemento con informacion de la conexion
DESCRIPCION: Se llama a esta funcion cuando el capturador recibe un SYN-ACK para
```

```
        obtener el numero de secuencia generado por la aplicación.
        busca en la hash del adaptador si existe una entrada para ese origen
        y destino, si es asi, le asigna el numero de secuencia del paquete a
        la hash para tener datos validos en los campos
*************************************************************************/
infoHash* getNumSecuence(HeaderPacket *pktSynAck){
   hashTable *auxHashAdaptador;
   infoHash * auxInfo;

   if (pktSynAck->usIdentification != Ident){ //Si es el propio paquete que acabo de mandar no lo tengo que tratar!!!!
      for (auxHashAdaptador = gHashAdaptador; auxHashAdaptador != NULL;auxHashAdaptador =
auxHashAdaptador->next){
         //aqui hay que comprobar que el SYN-ACK tiene la entrada que vamos a comprobar (que no es NULL la hash).
         if ((auxHashAdaptador->ipSource == pktSynAck->ulIPDestination) && (auxHashAdaptador->ptoSource ==
pktSynAck->usPortDestinationTCP)){
            if (auxHashAdaptador->info == NULL) {
               printf("Error, entrada en la hash del adaptador sin informacion en info\n");
               return NULL;
            }
            for (auxInfo = auxHashAdaptador->info; auxInfo != NULL; auxInfo = auxInfo->next) {
               if ((auxInfo->ipDest == pktSynAck->ulIPSource) && (auxInfo->ptoDest == pktSynAck->usPortSourceTCP))
{
                  auxInfo->numAsentimiento = pktSynAck->ulSecuenceNumber + 1;
                  auxInfo->numSecPkt = pktSynAck->ulAckNumber;
                  return auxInfo;
               }
            }
            return NULL;
         }
      }
   } //fin if
   return NULL;
}

/*************************************************************************
NOMBRE:     CheckAckAdaptador
SINOPSIS:   Recibe un Ack para el Adaptador y borra del buffer los pkt confirmados
PARAMETROS:  HeaderPacket * (entrada): ACK recibido
         int (salida): 1 si existe la conexion, 0 si no
DESCRIPCION: Busca la conexion (origen y destino) en la hash del Adaptador, si la
         encuentra borra del buffer (DeletePktBuffer) y devuelve 1. Si no
         encuentra la conexion devuelve 0.
*************************************************************************/

int CheckAckAdaptador(HeaderPacket *pktAck){
   hashTable *auxHashAdaptador;
   infoHash  *auxInfoHash;

   if (pktAck->usIdentification != Ident){
      for (auxHashAdaptador = gHashAdaptador; auxHashAdaptador != NULL; auxHashAdaptador =
auxHashAdaptador->next){
         if ((auxHashAdaptador->ipSource == pktAck->ulIPDestination) && (auxHashAdaptador->ptoSource == pktAck-
>usPortDestinationTCP )) {//&& (auxHashAdaptador->info->buffer != NULL)){
            for (auxInfoHash = auxHashAdaptador->info; auxInfoHash != NULL; auxInfoHash = auxInfoHash->next){
               if ((auxInfoHash->ipDest == pktAck->ulIPSource) && (auxInfoHash->ptoDest == pktAck-
>usPortSourceTCP)){
                  DeletePktBuffer (auxInfoHash,pktAck->ulAckNumber);
                  return 1;
               }
            }
            //Hemos encontrado la entrada en la hash origen, pero no en la de destino,
            //no tiene sentido seguir buscando en la origen
            return 0;
         }
      }
   }
   return 0;
}

/*************************************************************************
NOMBRE:     CreateSendSynAck
SINOPSIS:   Envia un Ack/SYN/SYN-ACK
PARAMETROS:  HeaderPacket * (entrada): pkt con informacion de la conexion
         unsigned long (entrada): num de secuencia del pkt a enviar
         int (entrada): informa de si el Flag SYN esta activo
```

```
        int (entrada): informa de si el Flag ACK esta activo
        bool (entrada):indica si hay que invertir Origen-destino
        AnsiString (entrada): Ruta de salida
DESCRIPCION: Envia un pkt SYN, SYN-ACK o ACK segun lo indicado en los parametros
        a la direccion indicada (Capturador/Redirector)
*****************************************************************************/
/*
  Crea un paquete ACK que asiente el paquete que recibe como parámetro
  - Recibe:
        - un paquete del que extraerá las IPs
        - el número de secuencia a poner en el paquete SYN
        - syn, será un SYN si syn=1
        - ack, será un ACK si ack=1
*/
void CreateSendSynAck (HeaderPacket *pkt, unsigned long numSec,int syn, int ack,bool cambiar,AnsiString  fileName){

   HeaderPacket *newPkt;

   newPkt         = (struct HeaderPacket*) calloc(1,sizeof(struct HeaderPacket));
   newPkt->usProto = (0x0800);
   // IP
   if (cambiar){ //Si es un ACK simulado, cambio IP origen y destino
      newPkt->ulIPSource          = pkt->ulIPDestination;
      newPkt->ulIPDestination     = pkt->ulIPSource;
      newPkt->usPortSourceTCP     = pkt->usPortDestinationTCP;
      newPkt->usPortDestinationTCP = pkt->usPortSourceTCP;
   }
   else{
      newPkt->ulIPSource          = pkt->ulIPSource;
      newPkt->ulIPDestination     = pkt->ulIPDestination;
      newPkt->usPortSourceTCP     = pkt->usPortSourceTCP;
      newPkt->usPortDestinationTCP = pkt->usPortDestinationTCP;
   }
   newPkt->ucVersion       = pkt->ucVersion;
   newPkt->ucLongHeader    = (unsigned char) 5; // si no hay opciones son 20 octetos
                             // que en palabras de 32 bits es 5
   newPkt->ucTypeServive   = pkt->ucTypeServive;
   newPkt->ucPrecedence    = (unsigned char) 0;
   newPkt->ucTOS           = (unsigned char) 0;
   newPkt->ucMBZ           = (unsigned char) 0;
   newPkt->usIdentification = (unsigned short) rand();
   newPkt->ucFlagReserved  = (unsigned char) 0;
   newPkt->ucFlagFragment  = (unsigned char) 0;
   newPkt->ucTTL           = pkt->ucTTL;
   newPkt->usHeaderChecsum  = (unsigned short) 0;
   newPkt->ucOptionsLenght  = (unsigned char)  0;
  /* TCP */
   newPkt->ucProtocol      = (unsigned char) 6;

   //Si es Syn-Ack o Fin tenemos que incrementar el número de ack
   if ((syn == 1 && ack == 1)){
      newPkt->ulAckNumber = pkt->ulSecuenceNumber + pkt->uiLengthData - (pkt->ucDataOffset*4) + 1;
   }
   else if(syn == 1){ //Si es sólo SYN tenemos que mandar ACK number a 0
      newPkt->ulAckNumber = 0;
   }
   else if (pkt->ucFsinc == 1){ //El paquete anterior es un SYN-ACK, por lo tanto contesto con ACK puro
      newPkt->ulAckNumber = pkt->ulSecuenceNumber + pkt->uiLengthData - (pkt->ucDataOffset*4) + 1;
   }
   else {
      newPkt->ulAckNumber = pkt->ulSecuenceNumber + pkt->uiLengthData - (pkt->ucDataOffset*4);
   }
   newPkt->ucDataOffset        = (unsigned char) 5;//5 si no hay opciones en la cabecera
   newPkt->ucFurg              = (unsigned char) 0;
   newPkt->ucFack              = (unsigned char) ack;
   newPkt->ucFpush             = (unsigned char) 0;
   newPkt->ucFrstConnect       = (unsigned char) 0;
   newPkt->ucFsinc             = (unsigned char) syn;
   newPkt->ucFdataEnd          = (unsigned char) 0;
   newPkt->usWin               = 4*MSS;  /* El más común */
   newPkt->usChecksunTCP       = (unsigned short) 0;
   newPkt->ulSecuenceNumber    = numSec;
   memset(newPkt->cData,0,MAX_BUFF_DATA);     /* Campo de Datos a 0 */
   newPkt->uiLengthData        = newPkt->ucDataOffset * 4; /* ACK sin datos */
   newPkt->shTotalLength       = 40 ;
```

```
   newPkt->ucFlagMoreFragment   = (unsigned char)0;
   newPkt->scFragmentOffset     = (unsigned short) 0;

   newPkt->cLatido              =  0;

   SendPkt(fileName,newPkt);
   free(newPkt);
}

/*****************************************************************************
NOMBRE:      CreateSendAck
SINOPSIS:    Envia un Ack
PARAMETROS:  HeaderPacket * (entrada)
             infoHash * (entrada)
             bool (entrada)
             AnsiString (entrada)
DESCRIPCION: Crea y envia  Ack con los datos del pkt pasado como parametro
*****************************************************************************/
void CreateSendAck (HeaderPacket *pkt, infoHash *info, bool cambiar, AnsiString fileName) {

   HeaderPacket *newPkt;

   newPkt         = (struct HeaderPacket*) calloc(1,sizeof(struct HeaderPacket));
   newPkt->usProto = (0x0800);
   // IP
   if (cambiar){ //Si es un ACK simulado, cambio IP origen y destino
      newPkt->ulIPSource          = pkt->ulIPDestination;
      newPkt->ulIPDestination     = pkt->ulIPSource;
      newPkt->usPortSourceTCP     = pkt->usPortDestinationTCP;
      newPkt->usPortDestinationTCP = pkt->usPortSourceTCP;
   }
   else{
      newPkt->ulIPSource          = pkt->ulIPSource;
      newPkt->ulIPDestination     = pkt->ulIPDestination;
      newPkt->usPortSourceTCP     = pkt->usPortSourceTCP;
      newPkt->usPortDestinationTCP = pkt->usPortDestinationTCP;
   }
   newPkt->ucVersion      = pkt->ucVersion;
   newPkt->ucLongHeader    = (unsigned char) 5; // si no hay opciones son 20 octetos
                             // que en palabras de 32 bits es 5
   newPkt->ucTypeServive    = pkt->ucTypeServive;
   newPkt->ucPrecedence     = (unsigned char) 0;
   newPkt->ucTOS            = (unsigned char) 0;
   newPkt->ucMBZ            = (unsigned char) 0;
   newPkt->usIdentification = (unsigned short) rand();
   newPkt->ucFlagReserved   = (unsigned char) 0;
   newPkt->ucFlagFragment   = (unsigned char) 0;
   newPkt->ucTTL            = pkt->ucTTL;
   newPkt->usHeaderChecsum  = (unsigned short) 0;
   newPkt->ucOptionsLenght  = (unsigned char)  0;
  /* TCP */
   newPkt->ucProtocol       = (unsigned char) 6;

   newPkt->ucDataOffset        = (unsigned char) 5;//5 si no hay opciones en la cabecera
   newPkt->ucFurg              = (unsigned char) 0;
   newPkt->ucFack              = (unsigned char) 1;
   newPkt->ucFpush             = (unsigned char) 0;
   newPkt->ucFrstConnect       = (unsigned char) 0;
   newPkt->ucFsinc             = (unsigned char) 0;
   newPkt->ucFdataEnd          = (unsigned char) 0;
   newPkt->usWin               = 4*MSS;  /* El más común */
   newPkt->usChecksunTCP       = (unsigned short) 0;
   memset(newPkt->cData,0,MAX_BUFF_DATA);      /* Campo de Datos a 0 */
   newPkt->uiLengthData        = newPkt->ucDataOffset * 4; /* ACK sin datos */
   newPkt->shTotalLength       = 40 ;
   newPkt->ucFlagMoreFragment   = (unsigned char) 0;
   newPkt->scFragmentOffset     = (unsigned short) 0;
   newPkt->ulAckNumber         = info->numSecPkt;
   newPkt->ulSecuenceNumber     = info->numAsentimiento;

   newPkt->cLatido          =  0;

   SendPkt(fileName,newPkt);
   free(newPkt);
}
```

```
/*****************************************************************************
NOMBRE:      CreateAckFIN
SINOPSIS:    Envia un Ack al Capturador
PARAMETROS:  HeaderPacket * (entrada)
             unsigned long (entrada)
DESCRIPCION: Crea y envia al Capturador un Ack de respuesta a un FIN
             Al ser de un FIN el num de Ack debe incrementarse en uno.
*****************************************************************************/
void CreateAckFin(HeaderPacket *pkt, unsigned long numSec){
   HeaderPacket *newPkt;

   newPkt         = (struct HeaderPacket*) calloc(1,sizeof(struct HeaderPacket));
   newPkt->usProto = (0x0800);
   newPkt->ulIPSource       = pkt->ulIPDestination;
   newPkt->ulIPDestination  = pkt->ulIPSource;
   newPkt->usPortSourceTCP      = pkt->usPortDestinationTCP;
   newPkt->usPortDestinationTCP = pkt->usPortSourceTCP;
   newPkt->ucVersion        = pkt->ucVersion;
   newPkt->ucLongHeader     = (unsigned char) 5; // si no hay opciones son 20 octetos
                              // que en palabras de 32 bits es 5
   newPkt->ucTypeServive    = (unsigned char) 0;
   newPkt->ucPrecedence     = (unsigned char) 0;
   newPkt->ucTOS            = (unsigned char) 0;
   newPkt->ucMBZ            = (unsigned char) 0;
   newPkt->usIdentification = (unsigned short) rand();
   newPkt->ucFlagReserved   = (unsigned char) 0;
   newPkt->ucFlagFragment   = (unsigned char) 0;
   newPkt->ucTTL            = pkt->ucTTL;
   newPkt->usHeaderChecsum  = (unsigned short) 0;
   newPkt->ucOptionsLenght  = (unsigned char)  0;
   /* TCP */
   newPkt->ucProtocol       = (unsigned char) 6;
   newPkt->ulAckNumber      = pkt->ulSecuenceNumber + pkt->uiLengthData - (pkt->ucDataOffset*4) + 1;
   newPkt->ucDataOffset     = (unsigned char) 5;//5 si no hay opciones en la cabecera
   newPkt->ucFurg           = (unsigned char) 0;
   newPkt->ucFack           = (unsigned char) 1;
   newPkt->ucFpush          = (unsigned char) 0;
   newPkt->ucFrstConnect    = (unsigned char) 0;
   newPkt->ucFsinc          = (unsigned char) 0;
   newPkt->ucFdataEnd       = (unsigned char) 0; //Tambien se podría poner a 1
   newPkt->usWin            = 4*MSS;  /* El más común */
   newPkt->usChecksunTCP    = (unsigned short) 0;
   newPkt->ulSecuenceNumber = numSec;
   memset(newPkt->cData,0,MAX_BUFF_DATA);      /* Campo de Datos a 0 */
   newPkt->uiLengthData     = newPkt->ucDataOffset * 4; /* ACK sin datos */
   newPkt->shTotalLength    = 40 ;
   newPkt->ucFlagMoreFragment = (unsigned char)0;
   newPkt->scFragmentOffset   = (unsigned short) 0;

   newPkt->cLatido           =  0;

   SendPkt(fileNameEnv,newPkt);
   free(newPkt);
}

/*****************************************************************************
NOMBRE:      CreateSendReset
SINOPSIS:    Envia un Reset
PARAMETROS:  HeaderPacket * (entrada)
             bool (entrada)
             AnsiString (entrada)
DESCRIPCION: Crea y envia un reset con los datos del pkt pasado como parametro
*****************************************************************************/
void CreateSendReset (HeaderPacket *pkt, bool cambiar, AnsiString fileName){
   HeaderPacket *newPkt;

   newPkt = (struct HeaderPacket*) calloc(1,sizeof(struct HeaderPacket));
   newPkt->usProto       = (0x0800);
   // IP
   if (cambiar){
      newPkt->ulIPSource          = pkt->ulIPDestination;
      newPkt->ulIPDestination     = pkt->ulIPSource;
      newPkt->usPortSourceTCP     = pkt->usPortDestinationTCP;
```

```
        newPkt->usPortDestinationTCP = pkt->usPortSourceTCP;
    }
    else{
        newPkt->ulIPSource          = pkt->ulIPSource;
        newPkt->ulIPDestination     = pkt->ulIPDestination;
        newPkt->usPortSourceTCP     = pkt->usPortSourceTCP;
        newPkt->usPortDestinationTCP = pkt->usPortDestinationTCP;
    }
    newPkt->ucVersion       = pkt->ucVersion;
    newPkt->ucLongHeader    = (unsigned char) 5; // si no hay opciones son 20 octetos
                              // que en palabras de 32 bits es 5
    newPkt->ucTypeServive   = pkt->ucTypeServive;
    newPkt->ucPrecedence    = (unsigned char) 0;
    newPkt->ucTOS           = (unsigned char) 0;
    newPkt->ucMBZ           = (unsigned char) 0;
    newPkt->usIdentification = (unsigned short) rand();
    newPkt->ucFlagReserved  = (unsigned char) 0;
    newPkt->ucFlagFragment  = (unsigned char) 0;
    newPkt->ucTTL           = pkt->ucTTL;
    newPkt->usHeaderChecsum = (unsigned short) 0;
    newPkt->ucOptionsLenght = (unsigned char)  0;
    /* TCP */
    newPkt->ucProtocol      = (unsigned char) 6;

    //Si es un reset a un Syn, el numero de secuencia es cero, sino el del ack
    if (pkt->ucFsinc == 1 && pkt->ucFack == 0) {
        newPkt->ulSecuenceNumber = 0;
    }
    else {
        newPkt->ulSecuenceNumber = pkt->ulAckNumber;
    }
    if (pkt->ucFsinc == 1) {
        newPkt->ulAckNumber = pkt->ulSecuenceNumber + 1;
    }
    else {
        newPkt->ulAckNumber = pkt->ulSecuenceNumber + pkt->uiLengthData - (pkt->ucDataOffset*4);
    }
    if (pkt->ucFdataEnd == 1){
        newPkt->ulAckNumber++;
    }

    newPkt->ucDataOffset        = (unsigned char) 5;//5 si no hay opciones en la cabecera
    newPkt->ucFurg              = (unsigned char) 0;
    newPkt->ucFack              = (unsigned char) 1;
    newPkt->ucFpush             = (unsigned char) 0;
    newPkt->ucFrstConnect       = (unsigned char) 1;
    newPkt->ucFsinc             = (unsigned char) 0;
    newPkt->ucFdataEnd          = (unsigned char) 0;
    newPkt->usWin               = 4*MSS;  /* El más común */
    newPkt->usChecksunTCP       = (unsigned short) 0;
    memset(newPkt->cData,0,MAX_BUFF_DATA);      /* Campo de Datos a 0 */
    newPkt->uiLengthData        = newPkt->ucDataOffset * 4; /* ACK sin datos */
    newPkt->shTotalLength       = 40 ;
    newPkt->ucFlagMoreFragment = (unsigned char)  0;
    newPkt->scFragmentOffset    = (unsigned short) 0;
    newPkt->usNumPacTotal       = 1;
    newPkt->usPacActual         = 1;

    newPkt->cLatido           =  0;

    SendPkt(fileName,newPkt);
    free(newPkt);
}

/*****************************************************************************
NOMBRE:      CreateSendResetDual
SINOPSIS:    Envia un Reset
PARAMETROS:  HeaderPacket * (entrada)
             infoHash * (entrada)
             AnsiString (entrada)
DESCRIPCION: Crea y envia un reset con los datos del pkt pasado como parametro y
             con los datos de la hash
*****************************************************************************/
void CreateSendResetDual(HeaderPacket*pkt, infoHash *info, AnsiString fileName){
    HeaderPacket *newPkt;
```

```
    newPkt = (struct HeaderPacket*) calloc(1,sizeof(struct HeaderPacket));
    newPkt->usProto        = (0x0800);
    // IP
    newPkt->ulIPSource          = pkt->ulIPSource;
    newPkt->ulIPDestination     = pkt->ulIPDestination;
    newPkt->usPortSourceTCP     = pkt->usPortSourceTCP;
    newPkt->usPortDestinationTCP = pkt->usPortDestinationTCP;

    newPkt->ucVersion       = pkt->ucVersion;
    newPkt->ucLongHeader    = (unsigned char) 5; // si no hay opciones son 20 octetos
                              // que en palabras de 32 bits es 5
    newPkt->ucTypeServive   = pkt->ucTypeServive;
    newPkt->ucPrecedence    = (unsigned char) 0;
    newPkt->ucTOS           = (unsigned char) 0;
    newPkt->ucMBZ           = (unsigned char) 0;
    newPkt->usIdentification = (unsigned short) rand();
    newPkt->ucFlagReserved  = (unsigned char) 0;
    newPkt->ucFlagFragment  = (unsigned char) 0;
    newPkt->ucTTL           = pkt->ucTTL;
    newPkt->usHeaderChecsum  = (unsigned short) 0;
    newPkt->ucOptionsLenght  = (unsigned char)  0;
    /* TCP */
    newPkt->ucProtocol      = (unsigned char) 6;

    //Si es un reset a un Syn, el numero de secuencia es cero, sino el del ack
    newPkt->ulSecuenceNumber = info->numAsentimiento;
    newPkt->ulAckNumber     = info->numSecPkt;

    newPkt->ucDataOffset        = (unsigned char) 5;//5 si no hay opciones en la cabecera
    newPkt->ucFurg              = (unsigned char) 0;
    newPkt->ucFack              = (unsigned char) 1;
    newPkt->ucFpush             = (unsigned char) 0;
    newPkt->ucFrstConnect       = (unsigned char) 1;
    newPkt->ucFsinc             = (unsigned char) 0;
    newPkt->ucFdataEnd          = (unsigned char) 0;
    newPkt->usWin               = 4*MSS;  /* El más común */
    newPkt->usChecksunTCP       = (unsigned short) 0;
    memset(newPkt->cData,0,MAX_BUFF_DATA);     /* Campo de Datos a 0 */
    newPkt->uiLengthData        = newPkt->ucDataOffset * 4; /* ACK sin datos */
    newPkt->shTotalLength       = 40 ;
    newPkt->ucFlagMoreFragment = (unsigned char)  0;
    newPkt->scFragmentOffset    = (unsigned short) 0;
    newPkt->usNumPacTotal       = 1;
    newPkt->usPacActual         = 1;

    newPkt->cLatido          =  0;

    SendPkt(fileName,newPkt);
    free(newPkt);
}

/*****************************************************************************
NOMBRE:      CreateSendResetHash
SINOPSIS:    Envia un Reset
PARAMETROS:  HeaderPacket * (entrada)
             bool (entrada)
             AnsiString (entrada)
DESCRIPCION:
*****************************************************************************/
void CreateSendResetHash (hashTable *origen, infoHash *destino, bool directo){
    HeaderPacket *newPkt;
    AnsiString fileName;

    newPkt = (struct HeaderPacket*) calloc(1,sizeof(struct HeaderPacket));
    newPkt->usProto        = (0x0800);
    // IP
    newPkt->ulIPSource          = destino->ipDest;
    newPkt->ulIPDestination     = origen->ipSource;
    newPkt->usPortSourceTCP     = destino->ptoDest;
    newPkt->usPortDestinationTCP = origen->ptoSource;

    if (directo){
       fileName = fileNameEnv;
    }
```

```
    else{
        printf("Enviando RST hash al redirector\n");
        fileName = fileNameRedirector;
    }
    newPkt->ucVersion       = '4';
    newPkt->ucLongHeader    = (unsigned char) 5; // si no hay opciones son 20 octetos
                              // que en palabras de 32 bits es 5
    newPkt->ucTypeServive   = (unsigned char) 0;
    newPkt->ucPrecedence    = (unsigned char) 0;
    newPkt->ucTOS           = (unsigned char) 0;
    newPkt->ucMBZ           = (unsigned char) 0;
    newPkt->usIdentification = (unsigned short) rand();
    newPkt->ucFlagReserved  = (unsigned char) 0;
    newPkt->ucFlagFragment  = (unsigned char) 0;
    newPkt->ucTTL           = (unsigned char) 0;
    newPkt->usHeaderChecsum  = (unsigned short) 0;
    newPkt->ucOptionsLenght  = (unsigned char)  0;
    /* TCP */
    newPkt->ucProtocol      = (unsigned char) 6;

    newPkt->ulSecuenceNumber = destino->numAsentimiento;
    newPkt->ulAckNumber     = destino->numSecPkt;

    newPkt->ucDataOffset        = (unsigned char) 5;//5 si no hay opciones en la cabecera
    newPkt->ucFurg              = (unsigned char) 0;
    newPkt->ucFack              = (unsigned char) 1;
    newPkt->ucFpush             = (unsigned char) 0;
    newPkt->ucFrstConnect       = (unsigned char) 1;
    newPkt->ucFsinc             = (unsigned char) 0;
    newPkt->ucFdataEnd          = (unsigned char) 0;
    newPkt->usWin               = 4*MSS;  /* El más común */
    newPkt->usChecksunTCP       = (unsigned short) 0;
    memset(newPkt->cData,0,MAX_BUFF_DATA);      /* Campo de Datos a 0 */
    newPkt->uiLengthData        = newPkt->ucDataOffset * 4; /* ACK sin datos */
    newPkt->shTotalLength       = 40 ;
    newPkt->ucFlagMoreFragment = (unsigned char)  0;
    newPkt->scFragmentOffset    = (unsigned short) 0;
    newPkt->usNumPacTotal       = 1;
    newPkt->usPacActual         = 1;

    newPkt->cLatido           = 0;

    SendPkt(fileName,newPkt);
    free(newPkt);
}


/*****************************************************************************
NOMBRE:      search_RTx
SINOPSIS:    Pide Rtx de los pkt que le faltan
PARAMETROS:  infoHash * (entrada): Hash con el buffer de pkt recibidos hasta el
                momento y la información del numero total de pkt a esperar.
DESCRIPCION: Segun la estructura de la comunicacion Sc-Sc, sabemos el numero de
            pkt que tenemos que recibir y su posición relativa. Esta función
            busca los pkt que aun no se han recibido y por cada uno de ellos
            envia un NACK (Sc-Sc) al redirector con el numero de pkt que falta.
*****************************************************************************/
void search_RTx(infoHash *info){

    bufferPkt *auxBuffer;
    int contador, total_pkt;

    auxBuffer = info->buffer;
    total_pkt = info->buffer->pkt.usNumPacTotal;

    for (contador = 1; contador < total_pkt + 1; contador++){
        if (auxBuffer == NULL) {
            //Mandar NACK SC_SC
            CreateSendAck_SC_SC(&(info->buffer->pkt), contador);
        }
        else if (auxBuffer->pkt.usPacActual == contador) {
            auxBuffer = auxBuffer->next;
        } else{
            //Mandar NACK comunicacion SC_SC
            CreateSendAck_SC_SC(&(auxBuffer->pkt), contador);
```

```
      }
   }
}


/*****************************************************************************
NOMBRE:      CreateSendAck_SC_SC
SINOPSIS:    Envia ACK/NACK para las comunicaciones SC-SC
PARAMETROS:  HeaderPacket * (entrada): pkt con la información sobre la conexion
             int (entrada): valor para el campo usPacActual
DESCRIPCION: Crea y envia al Redirector un pkt para la comunicacion Sc-Sc (ACK/NACK)
             Estos pkt se caracterizan por sus campos:
             - usNumPacTotal = 0
             - usPacActual   = nº no nulo. Indica una peticion de Rtx o un ACK total.
*****************************************************************************/

void CreateSendAck_SC_SC(HeaderPacket *pkt, int numPktRtx){

   HeaderPacket *newPkt;

    newPkt         = (struct HeaderPacket*) calloc(1,sizeof(struct HeaderPacket));
    newPkt->usProto = (0x0800);

    // IP: se cambia el orden porque el paquete se manda en sentido contrario.
    newPkt->ulIPSource          = pkt->ulIPDestination;
    newPkt->ulIPDestination     = pkt->ulIPSource;
    newPkt->usPortSourceTCP     = pkt->usPortDestinationTCP;
    newPkt->usPortDestinationTCP = pkt->usPortSourceTCP;
    newPkt->ucVersion       = pkt->ucVersion;
    newPkt->ucLongHeader    = (unsigned char) 5; // si no hay opciones son 20 octetos
                              // que en palabras de 32 bits es 5
    newPkt->ucTypeServive   = pkt->ucTypeServive;
    newPkt->ucPrecedence    = (unsigned char) 0;
    newPkt->ucTOS           = (unsigned char) 0;
    newPkt->ucMBZ           = (unsigned char) 0;

    newPkt->usIdentification = (unsigned short) rand();
    newPkt->ucFlagReserved  = (unsigned char) 0;
    newPkt->ucFlagFragment  = (unsigned char) 0;
    newPkt->ucTTL           = pkt->ucTTL;
    newPkt->usHeaderChecsum  = (unsigned short) 0;
    newPkt->ucOptionsLenght  = (unsigned char)  0;

    /* TCP */
    newPkt->ucProtocol      = (unsigned char) 6;

    newPkt->ulAckNumber     = 0;
    newPkt->ulSecuenceNumber = 0;

    newPkt->ucDataOffset        = (unsigned char) 5;//5 si no hay opciones en la cabecera
    newPkt->ucFurg              = (unsigned char) 0;
    newPkt->ucFack              = (unsigned char) 1;

    newPkt->ucFpush             = (unsigned char) 0;
    newPkt->ucFrstConnect       = (unsigned char) 0;
    newPkt->ucFsinc             = (unsigned char) 0;

    newPkt->ucFdataEnd          = (unsigned char) 0;
    newPkt->usWin               = 4*MSS;  /* El más común */
    newPkt->usChecksunTCP       = (unsigned short) 0;

    memset(newPkt->cData,0,MAX_BUFF_DATA);     /* Campo de Datos a 0 */

    newPkt->uiLengthData = newPkt->ucDataOffset * 4; /* ACK sin datos */

    newPkt->shTotalLength     = 40 ;

    newPkt->ucFlagMoreFragment = (unsigned char)  0;
    newPkt->scFragmentOffset   = (unsigned short) 0;

    newPkt->cLatido           = 0;

    //Comunicacion SC_SC
    newPkt->usNumPacTotal = 0;
    newPkt->usPacActual   = numPktRtx;
```

```
   printf("Enviando ACK sc-sc al redirector\n");
   SendPkt(fileNameRedirector,newPkt);
   MyStat->pktSndRed++;
   MyStat->bytSndRed = MyStat->bytSndRed + newPkt->uiLengthData - (newPkt->ucDataOffset * 4);
   MyStat->pktSndRedTCP++;
   MyStat->bytSndRedTCP = MyStat->bytSndRedTCP + newPkt->uiLengthData - (newPkt->ucDataOffset * 4);

   free(newPkt);
}


/*****************************************************************************
NOMBRE:     searchReverseConnection
SINOPSIS:   Busca una conexion en una hash dada que corresponda con la inversa
            del pkt pasado como parametro.
PARAMETROS:  HeaderPacket * (entrada): pkt con la informacion de la conexion a buscar
            hashTable * (entrada): Hash donde buscar la conexion
            infoHash * (salida): Elemento con la informacion sobre la conexion encontrada
DESCRIPCION: En la hashTable pasada como parametro, busca si existe una conexión
            para el pkt pasado tb como parametro, pero con el Origen y el
            destino invertidos.
            (Si el pkt va de A -> B, mira si en la hash existe una conexion que
            vaya de B -> A).
            Devuelve un puntero a la entrada de la hash destino si la encuentra,
            NULL en caso contrario.
*****************************************************************************/
infoHash *searchReverseConnection(HeaderPacket*pktTCP, hashTable * hash){

   hashTable *auxHash;
   infoHash  *auxInfo;

   //Buscamos IP Origen - Puerto Origen
   for (auxHash = hash; auxHash != NULL ; auxHash = auxHash->next){
      if (auxHash->ipSource == pktTCP->ulIPDestination  &&  auxHash->ptoSource == pktTCP-
>usPortDestinationTCP){
         //Buscamos IP Destino - Puerto Destino
         for (auxInfo = auxHash->info; auxInfo != NULL ; auxInfo=auxInfo->next){
            if (auxInfo->ipDest == pktTCP->ulIPSource && auxInfo->ptoDest == pktTCP->usPortSourceTCP){
                  printf("Conexion encontrada\n");
               return auxInfo;
            }
         }
         //Si hemos encontrado el origen pero no el destino, no tiene sentido seguir buscando
         printf("Conexion NO encontrada\n");
         return NULL;
      }
   }
   printf("Conexion NO encontrada\n");
   return NULL;
}


/*****************************************************************************
NOMBRE:     searchDirectConnection
SINOPSIS:   Busca una conexion en una hash dada que corresponda con el pkt
            pasado como parametro.
PARAMETROS:  HeaderPacket * (entrada): pkt con la informacion de la conexion a buscar
            hashTable * (entrada): Hash donde buscar la conexion
            infoHash * (salida): Elemento con la informacion sobre la conexion encontrada
DESCRIPCION: En la hashTable pasado como parametro, busca si existe una conexión
            para el pkt pasado tb como parametro, con el mismo origen y destino.
            (Si el pkt va de A -> B, mira si en la hash existe una conexion que
            vaya de A -> B).
            Devuelve un puntero a la entrada de la hash destino si la encuentra,
            NULL en caso contrario.
*****************************************************************************/

infoHash *searchDirectConnection(HeaderPacket*pktTCP, hashTable * hash){

   hashTable *auxHash;
   infoHash  *auxInfo;

   //Buscamos IP Origen - Puerto Origen
   for (auxHash = hash; auxHash != NULL ; auxHash = auxHash->next){
```

```
      if (auxHash->ipSource == pktTCP->ulIPSource  &&  auxHash->ptoSource == pktTCP->usPortSourceTCP){
         //Buscamos IP Destino - Puerto Destino
         for (auxInfo = auxHash->info; auxInfo != NULL ; auxInfo=auxInfo->next){
            if(auxInfo->ipDest == pktTCP->ulIPDestination && auxInfo->ptoDest == pktTCP->usPortDestinationTCP){
            printf("Conexion encontrada\n");
               return auxInfo;
            }
         }
         //Si hemos encontrado el origen pero no el destino, no tiene sentido seguir buscando
         printf("Conexion NO encontrada\n");
         return NULL;
      }
   }printf("Conexion NO encontrada\n");
   return NULL;
}


/*****************************************************************************
NOMBRE:      RtxPkt
SINOPSIS:    envia al Redirector el pkt con numero relativo pasado como argumento del
             buffer de la hash pasado como parametro.
PARAMETROS:  infoHash * (entrada): Hash con el buffer donde buscar el pkt a Rtx
             unsigned short (entrada): Posicion relativa del pkt dentro del buffer
DESCRIPCION: Recorre el buffer de la hash dada hasta encontrar un pkt con mumero
             relativo igual al valor pasado como parametro. Si lo encuentra lo
             envia al redirector (SendPkt)
*****************************************************************************/
void RtxPkt(infoHash* info, unsigned short numPktRtx) {

   bufferPkt *auxBuffer;
   HeaderPacket *pktRtx;

   auxBuffer = info->buffer;
   //Busco el paquete en el buffer
   for(auxBuffer = info->buffer; auxBuffer != NULL; auxBuffer = auxBuffer->next){
      if(auxBuffer->pkt.usPacActual == numPktRtx){
         pktRtx = &(auxBuffer->pkt);
         pktRtx->usNumPacTotal = info->totalPktAda;
         printf("Enviando Rtx pkt %d al redirector\n", auxBuffer->pkt.usPacActual);
         SendPkt(fileNameRedirector,pktRtx);
         MyStat->pktSndRed++;
         MyStat->bytSndRed = MyStat->bytSndRed + pktRtx->uiLengthData - (pktRtx->ucDataOffset * 4);
         MyStat->pktSndRedTCP++;
         MyStat->bytSndRedTCP = MyStat->bytSndRedTCP + pktRtx->uiLengthData - (pktRtx->ucDataOffset * 4);

         return;
      }
   }
}


/*****************************************************************************
NOMBRE:      CreateSendAckRst
SINOPSIS:    Envia ACK de un Rst recibido de otro Sc, para las comunicaciones SC-SC
PARAMETROS:  HeaderPacket * (entrada): pkt con la informacion sobre la conexion
DESCRIPCION: Crea y envia al Redirector un pkt para la comunicacion Sc-Sc (ACK
             de un Rst). confirma al otro Sc que hemos recibido su Reset.
             Estos pkt se caracterizan por sus campos:
             - usNumPacTotal = 1
             - usPacActual   = 0
*****************************************************************************/
void CreateSendAckReset (HeaderPacket *pkt){
   HeaderPacket *newPkt;

   newPkt                  = (struct HeaderPacket*) calloc(1,sizeof(struct HeaderPacket));
   newPkt->usProto             = (0x0800);

   // IP
   newPkt->ulIPSource          = pkt->ulIPDestination;
   newPkt->ulIPDestination     = pkt->ulIPSource;
   newPkt->usPortSourceTCP     = pkt->usPortDestinationTCP;
   newPkt->usPortDestinationTCP = pkt->usPortSourceTCP;

   newPkt->ucVersion           = pkt->ucVersion;
   newPkt->ucLongHeader        = (unsigned char) 5; // si no hay opciones son 20 octetos
```

```
                                    // que en palabras de 32 bits es 5
   newPkt->ucTypeServive        = pkt->ucTypeServive;
   newPkt->ucPrecedence         = (unsigned char) 0;
   newPkt->ucTOS                = (unsigned char) 0;
   newPkt->ucMBZ                = (unsigned char) 0;
   newPkt->usIdentification     = (unsigned short) rand();
   newPkt->ucFlagReserved       = (unsigned char) 0;
   newPkt->ucFlagFragment       = (unsigned char) 0;
   newPkt->ucTTL                = pkt->ucTTL;
   newPkt->usHeaderChecsum      = (unsigned short) 0;
   newPkt->ucOptionsLenght      = (unsigned char)  0;
   /* TCP */
   newPkt->ucProtocol           = (unsigned char) 6;

   newPkt->ulSecuenceNumber     = 0;
   newPkt->ulAckNumber          = pkt->ulSecuenceNumber + 1;

   newPkt->ucDataOffset         = (unsigned char) 5;//5 si no hay opciones en la cabecera
   newPkt->ucFurg               = (unsigned char) 0;
   newPkt->ucFack               = (unsigned char) 1;
   newPkt->ucFpush              = (unsigned char) 0;
   newPkt->ucFrstConnect        = (unsigned char) 0;
   newPkt->ucFsinc              = (unsigned char) 0;
   newPkt->ucFdataEnd           = (unsigned char) 0;
   newPkt->usWin                = 4*MSS;  /* El más común */
   newPkt->usChecksunTCP        = (unsigned short) 0;
   memset(newPkt->cData,0,MAX_BUFF_DATA);      /* Campo de Datos a 0 */
   newPkt->uiLengthData         = newPkt->ucDataOffset * 4; /* ACK sin datos */
   newPkt->shTotalLength        = 40 ;
   newPkt->ucFlagMoreFragment   = (unsigned char)  0;
   newPkt->scFragmentOffset     = (unsigned short) 0;

   // COMUNICACION SC-SC: Código del paquete ACK del RST
   newPkt->usNumPacTotal        = 1;
   newPkt->usPacActual          = 0;

   newPkt->cLatido              =  0;

   printf("Enviando ACK del Reset al redirector\n");
   SendPkt(fileNameRedirector,newPkt);
   MyStat->pktSndRed++;
   MyStat->bytSndRed = MyStat->bytSndRed + newPkt->uiLengthData - (newPkt->ucDataOffset * 4);
   MyStat->pktSndRedTCP++;
   MyStat->bytSndRedTCP = MyStat->bytSndRedTCP + newPkt->uiLengthData - (newPkt->ucDataOffset * 4);

   free(newPkt);
}


/*****************************************************************************
NOMBRE:      CreateSendAckWin0
SINOPSIS:    Envia un Ack al Capturador con un tamaño de ventana especifico
PARAMETROS:  HeaderPacket * (entrada): pkt con la informacion de la conexion
             unsigned long (entrada): valor del campo Numero de Secuencia
             unsigned long (entrada): Valor del campo Numero de Asentimiento
             unsigned short (entrada): Tamaño de la ventana
             bool (entrada): indica si hay que invertir origen-destino
DESCRIPCION: Crea y envia al Capturador un Ack con los datos (num sec, num ack,
             y tam ventana) dados. Se usa para la limitación por tamaño, para
             activar (win = 0) o desactivar (win = 4*MSS) la congestion.
*****************************************************************************/
void CreateSendAckWin0 (HeaderPacket *pkt,unsigned long numSec, unsigned long numAck, unsigned short win, bool
cambiar){
   HeaderPacket *newPkt;

   newPkt                    = (struct HeaderPacket*) calloc(1,sizeof(struct HeaderPacket));
   newPkt->usProto           = (0x0800);
   // IP
   if (cambiar) {
      newPkt->ulIPSource          = pkt->ulIPDestination;
      newPkt->ulIPDestination     = pkt->ulIPSource;
      newPkt->usPortSourceTCP     = pkt->usPortDestinationTCP;
      newPkt->usPortDestinationTCP = pkt->usPortSourceTCP;
   } else {
     newPkt->ulIPSource          = pkt->ulIPSource;
```

```
    newPkt->ulIPDestination       = pkt->ulIPDestination;
    newPkt->usPortSourceTCP       = pkt->usPortSourceTCP;
    newPkt->usPortDestinationTCP  = pkt->usPortDestinationTCP;
  }

  newPkt->ucVersion           = pkt->ucVersion;
  newPkt->ucLongHeader        = (unsigned char) 5; // si no hay opciones son 20 octetos
                                // que en palabras de 32 bits es 5
  newPkt->ucTypeServive       = pkt->ucTypeServive;
  newPkt->ucPrecedence        = (unsigned char) 0;
  newPkt->ucTOS               = (unsigned char) 0;
  newPkt->ucMBZ               = (unsigned char) 0;
  newPkt->usIdentification    = (unsigned short) rand();
  newPkt->ucFlagReserved      = (unsigned char) 0;
  newPkt->ucFlagFragment      = (unsigned char) 0;
  newPkt->ucTTL               = pkt->ucTTL;
  newPkt->usHeaderChecsum     = (unsigned short) 0;
  newPkt->ucOptionsLenght     = (unsigned char)  0;

  /* TCP */
  newPkt->ucProtocol          = (unsigned char) 6;
  newPkt->ulSecuenceNumber    = numSec;
  newPkt->ulAckNumber         = numAck;
  newPkt->ucDataOffset        = (unsigned char) 5;//5 si no hay opciones en la cabecera
  newPkt->ucFurg              = (unsigned char) 0;
  newPkt->ucFack              = (unsigned char) 1;
  newPkt->ucFpush             = (unsigned char) 0;
  newPkt->ucFrstConnect       = (unsigned char) 0;
  newPkt->ucFsinc             = (unsigned char) 0;
  newPkt->ucFdataEnd          = (unsigned char) 0;
  newPkt->usWin               = win;
  newPkt->usChecksunTCP       = (unsigned short) 0;
  memset(newPkt->cData,0,MAX_BUFF_DATA);     /* Campo de Datos a 0 */
  newPkt->uiLengthData        = newPkt->ucDataOffset * 4; /* ACK sin datos */
  newPkt->shTotalLength       = 40;
  newPkt->ucFlagMoreFragment  = (unsigned char)  0;
  newPkt->scFragmentOffset    = (unsigned short) 0;
  newPkt->usNumPacTotal       = 0;
  newPkt->usPacActual         = 0;

  newPkt->cLatido             =  0;

  SendPkt(fileNameEnv,newPkt);
  free(newPkt);
}


/*****************************************************************************
NOMBRE:      IntData
SINOPSIS:    Envia el buffer al Redirector (comunicacion Sc-Sc)
PARAMETROS:  MMRESULT (entrada)
             UINT (entrada)
             DWORD (entrada): Elemento con la informacion de la conexion y el
                      buffer a enviar
             DWORD (entrada)
             DWORD (entrada)
DESCRIPCION: Esta funcion se llama cuando ha vencido el timer de espera de pkt
             Aplicacion-Sc. Envia los paquetes almacenados en el buffer de la
             hash pasada como parametro al Redirector mediante la función
             SendBuffer2Redirector
*****************************************************************************/
void CALLBACK IntData(MMRESULT TimerID, UINT msg, DWORD ptrInfo, DWORD dw1, DWORD dw2){

  bufferPkt *auxBuffer;
  infoHash  *auxInfo;
  DWORD thId_Red = 0;

  printf("(IntData) Inicio\n");
  auxInfo   = (infoHash*) ptrInfo;
  auxBuffer = auxInfo->buffer;

  if (auxBuffer != NULL) {
     auxInfo->totalPktAda = auxInfo->totalPktCap;
     auxInfo->totalPktCap = 0;
```

```
        auxInfo->hiloRedirector =
CreateThread(NULL,0,(LPTHREAD_START_ROUTINE)SendBuffer2Redirector,auxInfo,0,&thId_Red);
        auxInfo->timerScSc = timeSetEvent(TIME_OVER_SC*(auxInfo->totalPktAda + 1), 0,int_SC_SC_Capturador,
(unsigned long)auxInfo, TIME_ONESHOT);

    } else {
        printf("Ha saltado la interrupcion pero no hay paquetes que enviar\n");
        printf("No hemos matado un timer correctamente, o lo hemos lanzado donde no debiamos\n");
        //system("PAUSE");
    }
}

/*****************************************************************************
NOMBRE:      IntSYN
SINOPSIS:    Envia un paquete de SYN al Redirector
PARAMETROS:  MMRESULT (entrada)
             UINT (entrada)
             DWORD (entrada): Elemento con la informacion de la conexion y el
                      pkt a enviar
             DWORD (entrada)
             DWORD (entrada)
DESCRIPCION: Esta funcion se llama cuando ha vencido el timer de espera de
             respuesta al SYN (Sc-Aplicacion). Envia al Redirector  un SYN con
             la información del primer pkt almacenado en el buffer de la hash
             pasada como parametro. Si se ha intentado mandar el SYN un numero
             mayor que el permitido por MAX_REINTENTOS se aborta la conexion.
*****************************************************************************/
void CALLBACK IntSYN(MMRESULT TimerID, UINT msg,DWORD ptrInfo, DWORD dw1, DWORD dw2){

    bufferPkt    *auxBuffer;
    infoHash     *auxInfo;
    HeaderPacket *pktTCP;
    DWORD dwWaitResult;

    printf("(IntSYN) Inicio\n");
    auxInfo   = (infoHash*) ptrInfo;
    auxBuffer = auxInfo->buffer;
    auxInfo->reintentosScSc++;

    if (auxInfo->reintentosScSc < MAX_REINTENTOS) {
        CreateSendSynAck(&(auxInfo->buffer->pkt),auxInfo->buffer->pkt.ulSecuenceNumber,1,0,false,fileNameEnv);
        auxInfo->timerID = timeSetEvent(TIMER_SYN, 0,IntSYN, (unsigned long)auxInfo, TIME_ONESHOT);
    }
    else {
        //Mandar RST al otro SrvCom
        auxInfo->rstRecibido = 1;
        if (auxInfo->buffer != NULL) {
            auxBuffer = auxInfo->buffer;
            while (auxBuffer->next != NULL) {
                auxBuffer = auxBuffer->next;
            }
            pktTCP = &(auxBuffer->pkt);
            printf("Interrupcion SYN, mandar RST al redirector\n");
            CreateSendReset(pktTCP, 1, fileNameRedirector);
            MyStat->pktSndRed++;
            MyStat->bytSndRed = MyStat->bytSndRed + pktTCP->uiLengthData - (pktTCP->ucDataOffset * 4);
            MyStat->pktSndRedTCP++;
            MyStat->bytSndRedTCP = MyStat->bytSndRedTCP + pktTCP->uiLengthData - (pktTCP->ucDataOffset * 4);

            auxInfo->reintentosScSc = 0;
            auxInfo->timerID = timeSetEvent((TIME_OVER_SC+1)*2, 0,IntRST, (unsigned long)auxInfo, TIME_ONESHOT);

            strcpy(MyAlarmas.cId ,"Discriminador");
            strcpy(MyAlarmas.cTxtAlarm,"[AVISO] Se ha superado el nº de reintentos, abortamos conexión\n");
            dwWaitResult = WaitForSingleObject(infoAlarmas.SemWrite,3000);
            switch(dwWaitResult){

                case WAIT_OBJECT_0:
                  memcpy(infoAlarmas.Memo,&MyAlarmas, sizeof(MyAlarmas));
                  ReleaseSemaphore(infoAlarmas.SemRead,1,NULL);
                  break;

                case WAIT_TIMEOUT:
                  printf("Time Out 1 hilo\n");
                  return;
```

```
            default:
              printf("failed 1 hilo\n");
              break;
          }//fin Switch
      }
   }
}

/*****************************************************************************
NOMBRE:      IntSYNACK
SINOPSIS:    Elimina una conexion semiabierta
PARAMETROS:  MMRESULT (entrada)
          UINT (entrada)
          DWORD (entrada): Elemento con la informacion de la conexion
          DWORD (entrada)
          DWORD (entrada)
DESCRIPCION: Esta funcion se llama cuando ha vencido el timer de espera de
          respuesta al SYN-ACK (Sc-Aplicacion). Elimina la conexion.
*****************************************************************************/
void CALLBACK IntSYNACK(MMRESULT TimerID, UINT msg,DWORD ptrInfo, DWORD dw1, DWORD dw2){

   infoHash     *auxInfo;
   HeaderPacket pktTCP;

   printf("(IntSYNACK) Inicio\n");
   auxInfo   = (infoHash*) ptrInfo;
   pktTCP    = auxInfo->buffer->pkt;

   //Borrar conexion
   DeleteConectionCapturador(pktTCP.ulIPSource, pktTCP.usPortSourceTCP, pktTCP.ulIPDestination,
pktTCP.usPortDestinationTCP);
}

/*****************************************************************************
NOMBRE:      IntFIN
SINOPSIS:    Elimina una conexión de las hash del Capturador y del Adpatador
PARAMETROS:  MMRESULT (entrada)
          UINT (entrada)
          DWORD (entrada): Elemento con la informacion de la conexion a borrar
          DWORD (entrada)
          DWORD (entrada)
DESCRIPCION: Esta funcion se llama cuando ha vencido el timer de cierre de
          conexiones. Recibe como parametro un puntero a una estructura con
          la informacion sobre la direcion IP y puerto Origen y Destino. Borra
          de la hash del Capturador y de la hash del adaptador la conexión
          que corresponda con los datos pasados.
          Siempre se llama con los datos referidos a la hash delCapturador
*****************************************************************************/
void CALLBACK IntFIN(MMRESULT TimerID, UINT msg,DWORD ptrInfo, DWORD dw1, DWORD dw2){

   openConnection *auxFin;
   unsigned long IpOrigen, IpDestino;
   unsigned short portOrigen, portDestino;

   printf("(IntFIN) Inicio\n");
   auxFin   = (openConnection*) ptrInfo;

   IpOrigen    = auxFin->ipSource;
   IpDestino   = auxFin->ipDest;
   portOrigen  = auxFin->ptoSource;
   portDestino = auxFin->ptoDest;
   //Doy por sentado que siempre se llama desde la hash del Capturador
   DeleteConectionCapturador (IpOrigen,portOrigen,IpDestino,portDestino);
   DeleteConectionAdaptador  (IpDestino,portDestino,IpOrigen,portOrigen);
   free(auxFin);
}


/*****************************************************************************
NOMBRE:      IntRST
SINOPSIS:    Elimina una conexión de las hash del Capturador y del Adpatador
PARAMETROS:  MMRESULT (entrada)
          UINT (entrada)
          DWORD (entrada): Elemento con la informacion de la conexion a borrar
```

```
            DWORD (entrada)
            DWORD (entrada)
DESCRIPCION: Esta funcion se llama cuando ha vencido el timer de espera de
            respuesta a un RST. Recibe como parametro un puntero a una
            estructura con la informacion sobre la direcion IP y puerto Origen
            y Destino. Borra de la hash del Capturador y de la hash del
            adaptador la conexión que corresponda con los datos pasados.
*****************************************************************************/
void CALLBACK IntRST(MMRESULT TimerID, UINT msg,DWORD ptrInfo, DWORD dw1, DWORD dw2){

   unsigned long IpOrigen, IpDestino;
   unsigned short portOrigen, portDestino;
   infoHash  *auxInfo;
   bufferPkt *auxBuffer;
   HeaderPacket *pktTCP;

   printf("Interrupcion RST\n");
   auxInfo   = (infoHash*) ptrInfo;
   auxInfo->reintentosScSc++;
   if (auxInfo->buffer != NULL) {
      auxBuffer = auxInfo->buffer;
      while (auxBuffer->next != NULL) {
         auxBuffer = auxBuffer->next;
      }
      pktTCP = &(auxBuffer->pkt);
   }
   else {
      //error
      return;
   }

   if (auxInfo->reintentosScSc < MAX_REINTENTOS) {
      if (auxInfo->conexDual != NULL) {
         printf("Interrupcion RST, mandar RST al redirector\n");
         CreateSendReset(pktTCP, 0, fileNameRedirector);
         MyStat->pktSndRed++;
         MyStat->bytSndRed = MyStat->bytSndRed + pktTCP->uiLengthData - (pktTCP->ucDataOffset * 4);
         MyStat->pktSndRedTCP++;
         MyStat->bytSndRedTCP = MyStat->bytSndRedTCP + pktTCP->uiLengthData - (pktTCP->ucDataOffset * 4);
      }
      else {
         printf("Interrupcion SYN, mandar RST al redirector\n");
         CreateSendReset(pktTCP, 1, fileNameRedirector);
         MyStat->pktSndRed++;
         MyStat->bytSndRed = MyStat->bytSndRed + pktTCP->uiLengthData - (pktTCP->ucDataOffset * 4);
         MyStat->pktSndRedTCP++;
         MyStat->bytSndRedTCP = MyStat->bytSndRedTCP + pktTCP->uiLengthData - (pktTCP->ucDataOffset * 4);
      }
      auxInfo->timerID = timeSetEvent((TIME_OVER_SC+1)*2, 0,IntRST, (unsigned long)auxInfo, TIME_ONESHOT);
   }
   else {
      IpOrigen    = pktTCP->ulIPSource;
      IpDestino   = pktTCP->ulIPDestination;
      portOrigen  = pktTCP->usPortSourceTCP;
      portDestino = pktTCP->usPortDestinationTCP;

      if (auxInfo->conexDual != NULL) {
         DeleteConectionCapturador (IpOrigen,portOrigen,IpDestino,portDestino);
         DeleteConectionAdaptador  (IpDestino,portDestino,IpOrigen,portOrigen);
      }
      else {
         DeleteConectionAdaptador (IpOrigen,portOrigen,IpDestino,portDestino);
      }
   }
}



/*****************************************************************************
NOMBRE:      int_SC_SC_Capturador
SINOPSIS:    Envia el ultimo pkt del buffer al Redirector (comunicacion Sc-Sc)
PARAMETROS:  MMRESULT (entrada)
            UINT (entrada)
            DWORD (entrada): Elemento con la informacion de la conexion y el
                        buffer a enviar
```

```
            DWORD (entrada)
            DWORD (entrada)
DESCRIPCION: Esta funcion se llama cuando ha vencido el timer de espera de
            respuesta del otro ServCom. Envia el ultimo paquete almacenado en
            el buffer y vuelve a esperar respuesta. Si superamos el numero de
            reintentos se aborta la conexion.
*****************************************************************************/
void CALLBACK int_SC_SC_Capturador(MMRESULT TimerID, UINT msg,DWORD ptrInfo, DWORD dw1, DWORD
dw2){

   bufferPkt *auxBuffer;
   infoHash  *auxInfo, *auxDual;
   HeaderPacket *pktTCP;
   unsigned long IpOrigen, IpDestino;
   unsigned short portOrigen, portDestino;
   DWORD dwWaitResult;


   printf("(int_SC_SC_Capturador) Inicio\n");
   auxInfo   = (infoHash*) ptrInfo;

   if (auxInfo->rstRecibido == 1) {
      printf("Caso extraño, salta la interrupcion antees de que me de tiempo a borrarla\n");
      return;
   }
   auxInfo->reintentosScSc++;
   if (auxInfo->reintentosScSc > MAX_REINTENTOS) {
      printf("(int_SC_SC_Capturador) superado el numero de reintentos. Abotamos conexion\n");
      // Mandar reset al capturador
      if (auxInfo->buffer != NULL){ //Solo mando un reset si puedo coger su referencia del Buffer
         auxBuffer = auxInfo->buffer;
         while (auxBuffer->next != NULL) {
            auxBuffer = auxBuffer->next;
         }
         pktTCP = &(auxBuffer->pkt);
         CreateSendReset (pktTCP, 1, fileNameEnv);
         IpOrigen    = pktTCP->ulIPSource;
         IpDestino   = pktTCP->ulIPDestination;
         portOrigen  = pktTCP->usPortSourceTCP;
         portDestino = pktTCP->usPortDestinationTCP;

         strcpy(MyAlarmas.cId ,"Discriminador");
         strcpy(MyAlarmas.cTxtAlarm,"[AVISO] Se ha superado el nº de reintentos, abortamos conexión\n");
         dwWaitResult = WaitForSingleObject(infoAlarmas.SemWrite,3000);
         switch(dwWaitResult){

            case WAIT_OBJECT_0:
               memcpy(infoAlarmas.Memo,&MyAlarmas, sizeof(MyAlarmas));
               ReleaseSemaphore(infoAlarmas.SemRead,1,NULL);
               break;

            case WAIT_TIMEOUT:
               printf("Time Out 1 hilo\n");
               return;

            default:
               printf("failed 1 hilo\n");
               break;
         }//fin Switch

         // Borrar conexiones
         auxDual = auxInfo->conexDual;
         DeleteConectionCapturador(IpOrigen, portOrigen, IpDestino, portDestino);
         if (auxDual != NULL) {
            DeleteConectionAdaptador (IpDestino, portDestino, IpOrigen, portOrigen);
         }
      }
      return;
   }
   else{
      printf("Interrupcion SC-SC Capturador, RTX\n");
      RtxPkt(auxInfo, auxInfo->totalPktAda);
      //Volver a activar el Timer
      auxInfo->timerScSc = timeSetEvent(TIME_OVER_SC*(auxInfo->totalPktAda + 1)*2, 0,int_SC_SC_Capturador,
(unsigned long)auxInfo, TIME_ONESHOT);
```

```
    }
}

/*****************************************************************************
NOMBRE:      int_SC_SC_Adaptador
SINOPSIS:    Solicita Rtx de los pkt que no han llegado
PARAMETROS:  MMRESULT (entrada)
             UINT (entrada)
             DWORD (entrada): Elemento con la informacion de la conexion y el
                      buffer con los pkt ya recibidos
             DWORD (entrada)
             DWORD (entrada)
DESCRIPCION: Esta funcion se llama cuando ha vencido el timer de espera de
             pkt del otro ServCom. Llama a la funcion search_RTx para que pida
             Rtx al otro Sc de los paquetes que faltan en el buffer. Si se ha
             superado el numero de reintentos se aborta la conexion.
*****************************************************************************/
void CALLBACK int_SC_SC_Adaptador(MMRESULT TimerID, UINT msg,DWORD ptrInfo, DWORD dw1, DWORD
dw2){
   infoHash  *auxInfo, *auxDual;
   HeaderPacket *pktTCP;
   bufferPkt *auxBuffer;
   unsigned long IpOrigen, IpDestino;
   unsigned short portOrigen, portDestino;
   DWORD dwWaitResult;

   printf("(int_SC_SC_Adaptador) Inicio\n");
   auxInfo   = (infoHash*) ptrInfo;

   if (auxInfo->rstRecibido == 1) {
      printf("Caso extraño, salta la interrupcion antees de que me de tiempo a borrarla\n");
      return;
   }
   auxInfo->reintentosScSc++;
   if (auxInfo->reintentosScSc > MAX_REINTENTOS) {
      printf("(int_SC_SC_Adaptador) superado el numero de reintentos. Abotamos conexion\n");
      if (auxInfo->buffer != NULL) {
         auxBuffer = auxInfo->buffer;
         while (auxBuffer->next != NULL) {
            auxBuffer = auxBuffer->next;
         }
         pktTCP = &(auxBuffer->pkt);
         auxDual = auxInfo->conexDual;
         if (auxDual != NULL) {
            CreateSendReset (pktTCP, 0, fileNameEnv);
         }
         IpOrigen    = pktTCP->ulIPSource;
         IpDestino   = pktTCP->ulIPDestination;
         portOrigen  = pktTCP->usPortSourceTCP;
         portDestino = pktTCP->usPortDestinationTCP;
         //Avisamos al supervisor
         strcpy(MyAlarmas.cId ,"Discriminador");
         strcpy(MyAlarmas.cTxtAlarm,"[AVISO] Se ha superado el nº de reintentos, abortamos conexión\n");
         dwWaitResult = WaitForSingleObject(infoAlarmas.SemWrite,3000);
         switch(dwWaitResult){

            case WAIT_OBJECT_0:
              memcpy(infoAlarmas.Memo,&MyAlarmas, sizeof(MyAlarmas));
              ReleaseSemaphore(infoAlarmas.SemRead,1,NULL);
              break;

            case WAIT_TIMEOUT:
              printf("Time Out 1 hilo\n");
              return;

            default:
              printf("failed 1 hilo\n");
              break;
         }//fin Switch

         //borrar conexiones
         DeleteConectionAdaptador(IpOrigen, portOrigen, IpDestino, portDestino);
         if (auxDual != NULL) {
            DeleteConectionCapturador (IpDestino, portDestino, IpOrigen, portOrigen);
         }
```

```
      }
      return;
   }

   //Busqueda de los paquetes que faltan y envio de nack
   if (auxInfo->buffer == NULL) {
      printf("Error, estamos esperando pkt y el buffer esta vacio\n");
      return;
   }
   search_RTx(auxInfo);

   //Volver a activar el Timer
   //auxInfo->timerScSc = timeSetEvent((TIME_OVER_SC+1)*2, 0,int_SC_SC_Adaptador, (unsigned long)auxInfo,
TIME_ONESHOT);
   auxInfo->timerScSc = timeSetEvent(auxInfo->vCanal, 0,int_SC_SC_Adaptador, (unsigned long)auxInfo,
TIME_ONESHOT);
}
```

9. Bibliografía

DOCUMENTOS IMPRESOS

Tanenbaum, Andrew S. (1997). *Redes de computadoras* (3ª ed). Prentice-Hall Hispanoamericana, S.A.

Stevens, Richard (1994). *TCP/IP Illustrated, Volume 1: The Protocols*. Addison-Wesley.

RECURSOS ELECTRÓNICOS

RFC 793 – *Transmission Control Protocol*
http://www.ietf.org/rfc/rfc793.txt
http://www.rfc-es.org/rfc/rfc0793-es.txt (en español)

RFC 1122 – *Requirements for Internet Hosts*
http://www.ietf.org/rfc/rfc1122.txt

RFC 1180 - *A TCP/IP Tutorial*
http://www.ietf.org/rfc/rfc1180.txt

WinPcap
http://www.winpcap.org/

Iperf
http://iperf.sourceforge.net/